GammaLoop

Differential cross-sections with Local Unitarity

1 Overview

GammaLoop is alpha-stage research software that computes numerical amplitude integrands and uses Local Unitarity for differential partonic cross sections. Its current external-kinematics path uses fixed momenta and a unit PDF factor; it does not yet provide hadronic PDF convolution or a factorization scale. Begin with the physics scope and method before translating a desired collider observable into a run card.

The primary manual path is written for perturbative-QFT and collider physicists. GammaLoop is an application rather than a thin numerical library: model import, process generation, integrand construction, integration, observables, persistence, and diagnostics meet in one stateful workflow. Rust and Python details become relevant only when structured automation is needed.

> **Primary interface**
> The command-line interface is the primary user interface. A run card creates a state, executes commands, and leaves a reusable working directory. The Rust and Python APIs load the same persisted state and expose selected structured operations; they do not define a second, independent execution model.

1.1 Choose a task

- To build or install the CLI and evaluate a compact one-loop scalar graph, follow the command-line guide.
- To decide whether an initial state, contribution, mass/spin choice, subtraction, and observable are supported, read physics scope and Local Unitarity.
- To define momentum flow, helicity and color ownership, flux, units, and sample weights, use the scientific conventions.
- To follow a maintained run card through the complete persistence lifecycle, use the first-state tutorial.
- To adapt a process specification or generation filters, use the process-generation guide with the generated `generate` reference.
- To inspect concrete cut events, selectors, and histogram snapshots without a full integration, follow the events and observables guide.
- To automate a loaded state or diagnose a long run, choose the Rust/Python interface guide or logging and diagnostics guide.

1.2 A stateful run

A typical source checkout is built and exercised from the repository root:

```
just build-cli-release
./gammaloop ./examples/cli/gg_hhh/1L/gg_hhh_1L.toml
```

The run card imports a model, generates the requested process, executes its command blocks, and records the resulting state. Resume that directory explicitly for subsequent work:

```
./gammaloop -s ./examples/cli/gg_hhh/1L/state \
  run integrate_physical -c "quit -o"
```

The persisted `run.toml` records how to replay the run. The settings files describe global and default runtime configuration, while `processes/` contains process-specific state. Ordinary runs create `gammaloop_state/` unless a different state path is supplied.

> **A lifecycle example is not a physics benchmark**
> The maintained `gg_hhh` card is useful because it exercises model loading, generation, integration settings, and persistence. It selects a fixed-helicity amplitude contribution with an explicit color projector; running it does not by itself produce an unpolarized, color-averaged cross section or validate a published observable.

1.3 Installation and external tools

For ordinary CLI use, begin with using GammaLoop from the command line. It recommends the packaged Nix application when Nix is available, but does not require it: a copy-and-paste Cargo route is provided for a standard Rust toolchain. Neither installation path requires a source checkout.

A source checkout is still needed to change GammaLoop or to work through repository-owned run cards. Its supported development path is the repository's Nix shell, or a local Rust toolchain together with `just`, a recent GNU toolchain, and Python 3.11 or newer when building bindings. FORM 4.2.1 or newer is needed for analytical integration of integrated UV counterterms. UFO model import also needs the Python `ufo-model-loader` package.

Diagram rendering is a separate concern and uses Clinnet and Typst. Building the CLI does not imply that these drawing tools are installed. Use the Clinnet DOT-rendering guide for their independent versions, templates, and cache boundary.

1.4 Related crates

> **How the crates fit together**
> Linnet provides the half-edge graph model and graph algorithms. Spenso provides typed tensors, tensor structures, and network execution. Idenso handles symbolic tensor identities and algebraic simplification. Vakint matches and evaluates vacuum-integral topologies. GammaLoop combines these components into a collider-calculation workflow.

For a deeper view of how the command layer, state, integrands, and external components fit together, see the GammaLoop implementation architecture.

1.5 Where to begin

- Use built-in `--help` for the CLI options supported by your installed version.
- Start with the included `gg_hhh` run card for a complete state lifecycle, not as a substitute for a physical normalization and validation plan.
- Record the method scope and every factor from the normalization checklist before launching a production run.
- Use the Rust or Python API only when a program needs structured access to loaded state or sample-evaluation results.
- Check prerequisites before starting an expensive integration; some workflows require a Symbolica license or external tools such as FORM.

2 Your first GammaLoop calculation

Generate the same scalar one-loop bubble through the interface that fits your work. Every route uses GammaLoop's built-in scalar model, disables integrated UV generation, and evaluates the same three-dimensional loop-momentum point. The result is a compact software check rather than a normalized collider prediction.

Command line
Use the executable for run cards, interactive commands, persistent states, and complete calculations. This is the shortest route for most GammaLoop users.

Use GammaLoop from the command line →

Python
Use the stateful Python API from notebooks, scripts, or workflow automation. It drives the same command session and returns structured evaluation records.

Use GammaLoop from Python →

Rust
Embed GammaLoop when a Rust application must own the state lifecycle, issue commands, or retain typed evaluation results without crossing a Python boundary.

Use GammaLoop from Rust →

If you are unsure, begin with the command-line guide. The interface guide explains the longer-term ownership and precision boundaries once the first calculation works.

3 Using GammaLoop from the command line

This guide builds one scalar one-loop topology, saves it as a GammaLoop state, and evaluates one point. It uses a built-in model and does not require a run card, a UFO model, or FORM. Once the executable is available, the example itself takes only a few seconds.

3.1 Get the CLI

> **Nix is recommended, not required**
> If you already use Nix, or want the most reproducible installation, follow the first route. Otherwise skip directly to the Cargo route below. Both routes provide the same `gammaloop` command, and neither requires a GammaLoop source checkout.

3.1.1 Recommended: the Nix package

The repository flake exposes GammaLoop as a packaged application. Install that package directly; a source checkout and development shell are not required:

```
nix profile add github:alphal00p/gammaloop#gammaloop
gammaloop --help
```

Nix downloads the package from a configured binary cache when it is available and otherwise builds the same derivation locally. The flake currently supports x86-64 and ARM64 Linux, plus Apple-silicon macOS.

To try the same packaged CLI without adding it to your profile, run:

```
nix run github:alphal00p/gammaloop#gammaloop -- --help
```

Use `nix run .#gammaloop -- --help` instead when testing the package from an existing source checkout. `nix develop` followed by `just build-cli` remains useful for contributors changing GammaLoop, but it is not the normal installation path.

3.1.2 No Nix: install with Cargo

Install a stable Rust toolchain with rustup, then install the small set of native build tools used by GammaLoop. On Debian or Ubuntu:

```
sudo apt-get update
sudo apt-get install -y build-essential m4 diffutils
```

On macOS, install Apple's command-line build tools if they are absent:

```
xcode-select --install
```

Then install the `gammaloop-api` crate. Its executable is named `gammaloop`:

```
SYMBOLICA_OEM_LICENSE=SYMBOLICA_OEM_GAMMALOOP \
  cargo install --locked gammaloop-api
gammaloop --help
```

This registry command becomes available with the first automated crates.io release. Until then, install the release candidate directly from its tested Git revision, without making a checkout:

```
SYMBOLICA_OEM_LICENSE=SYMBOLICA_OEM_GAMMALOOP \
  cargo install --locked \
  --git https://github.com/alphal00p/gammaloop.git \
  --rev eb1dab23c24345def88226ef10874f0dde6c4aa5 \
  gammaloop-api
```

Cargo compiles the CLI locally, so its first installation takes longer than downloading a cached Nix build. Its default build vendors the required GMP, MPFR, and MPC sources; OpenSSL, Python, FORM, and system copies of those libraries are not prerequisites for this example. On Windows, use WSL 2 and follow the Debian or Ubuntu instructions above. The `SYMBOLICA_OEM_LICENSE` value is GammaLoop's public build selector, not a private license key. Add the `ufo_support` feature and Python development headers only when external UFO model loading is needed. Symbolica's license terms still apply to both installation paths.

> **Cargo install is Nix-free, but it is still a source build**
> The current release workflow publishes crate sources and Python distributions, but not target-specific GammaLoop CLI archives or Cargo Binstall metadata. `cargo binstall` therefore cannot yet provide a supported prebuilt binary and may only fall back to a source build.

3.2 Generate a one-loop graph

Users who installed the Nix package or Cargo crate can run this command as written. Contributors using `just build-cli` in a source checkout should replace `gammaloop` with `./gammaloop`:

> **Choose a disposable state path**
> `--clean-state` removes and replaces the state directory passed with `-s`. Use the fresh `./gammaloop_state/quickstart` path shown below, or substitute another path whose existing contents you are prepared to delete.

```
gammaloop --clean-state -s ./gammaloop_state/quickstart run -c \
  'import model scalars-default.json; set global kv
global.generation.uv.generate_integrated=false; generate amp scalar_1 > scalar_1 [{1}] --
allowed-vertex-interactions V_3_SCALAR_122 -p bubble -i one_loop; display processes; quit -
o'
```

The command imports the built-in scalar model, generates one one-loop bubble amplitude, prints the process named `bubble` and integrand named `one_loop`, and writes a reusable state under `gammaloop_state/quickstart/`. Integrated UV generation is disabled deliberately so that this first topology does not invoke FORM.

Check the result by looking for `bubble` and `one_loop` in the process listing and for `gammaloop_state/quickstart/state_manifest.toml`. This graph is a compact software exercise, not a normalized collider prediction.

3.3 Evaluate one point

Load the state read-only and inspect the integrand at one loop-momentum point:

```
gammaloop --read-only-state -s ./gammaloop_state/quickstart run -c \
  'inspect -p bubble -i one_loop -x 0.1 0.2 0.3; quit'
```

A successful run prints the parameterization Jacobian and a finite complex value without changing the saved state. Continue with the first-state tutorial to use a maintained run card, inspect the complete persistence lifecycle, and prepare a calculation whose physics scope and normalization are explicit.

4 Using GammaLoop from Python

This route generates the same scalar bubble as the command-line example, but keeps the session in a Python object and returns a structured evaluation result. Use Python 3.11 or newer.

4.1 Install the Python package

Create an isolated environment. The ordinary wheel command targets the automated `0.3.4` release and newer:

```
python3.11 -m venv .venv
. .venv/bin/activate
python -m pip install --upgrade pip
python -m pip install "gammaloop>=0.3.4"
```

Until that release is visible on PyPI, build the current package from a source checkout instead. The public build selector is not a private license key:

```
git clone https://github.com/alphal00p/gammaloop.git
cd gammaloop
python3.11 -m venv .venv
. .venv/bin/activate
SYMBOLICA_OEM_LICENSE=SYMBOLICA_OEM_GAMMALOOP \
  python -m pip install .
```

4.2 Generate and evaluate

Save this as `first_gammaloop.py`:

```
from pathlib import Path
from tempfile import TemporaryDirectory

from gammaloop import GammaLoopAPI

with TemporaryDirectory() as temporary:
    api = GammaLoopAPI(state_folder=Path(temporary) / "state")
    api.run("import model scalars-default.json")
    api.run(
        "set global kv "
```

```
        "global.generation.uv.generate_integrated=false"
    )
    api.run(
        "generate amp scalar_1 > scalar_1 [{1}] "
        "--allowed-vertex-interactions V_3_SCALAR_122 "
        "-p bubble -i one_loop"
    )

    result = api.evaluate_sample(
        [0.1, 0.2, 0.3],
        process_id=0,
        integrand_name="one_loop",
    )
    print(result.integrand_result)
```

Run `python first_gammaloop.py`. A successful execution prints a finite complex value. The temporary state is deleted when the script exits; use a normal directory when the state should be persisted and explicitly execute a save command before closing the session.

> **One session owns one mutable state**
> `GammaLoopAPI.run` acts on the same in-memory model, settings, processes, and history as the structured methods. The object does not save automatically. Use separate instances for independent calculations, and inspect the interface guide before building a long-running workflow around mutable state.

The documentation harness syntax-checks this script. Runtime execution additionally needs the native wheel or source build and a Symbolica license mode appropriate for the user.

5 Using GammaLoop from Rust

Embed GammaLoop when a Rust program should own the state, command session, and typed evaluation result directly. This example generates the same scalar bubble as the CLI and Python routes.

5.1 Create a project

The ordinary crates.io command becomes available with the automated `0.3.4` release:

```
cargo new first-gammaloop
cd first-gammaloop
cargo add gammaloop-api@0.3.4 ndarray@0.17 eyre@0.6
```

Before that release, replace the `gammaloop-api` line with the tested release revision:

```
cargo add gammaloop-api \
  --git https://github.com/alphal00p/gammaloop.git \
  --rev eb1dab23c24345def88226ef10874f0dde6c4aa5
cargo add ndarray@0.17 eyre@0.6
```

5.2 Drive one state and evaluate it

Replace `src/main.rs` with:

```
use eyre::Result;
use gammaloop_api::{
    commands::evaluate_samples::{evaluate_sample, EvaluateSamples},
    state::CommandHistory,
```

```
    StateLoadOption,
};
use ndarray::arr2;
use std::time::{SystemTime, UNIX_EPOCH};

fn main() -> Result<()> {
    let nonce = SystemTime::now().duration_since(UNIX_EPOCH)?.as_nanos();
    let state_folder = std::env::temp_dir().join(format!("gammaloop-quickstart-{nonce}"));
    let mut loaded = StateLoadOption {
        state_folder: Some(state_folder.clone()),
        ..StateLoadOption::default()
    }
    .load()?;

    for raw in [
        "import model scalars-default.json",
        "set global kv global.generation.uv.generate_integrated=false",
        concat!(
            "generate amp scalar_1 > scalar_1 [{1}] ",
            "--allowed-vertex-interactions V_3_SCALAR_122 ",
            "-p bubble -i one_loop"
        ),
    ] {
        let command = CommandHistory::from_raw_string(raw)?;
        let _ = loaded.cli_session().execute_command(command)?;
    }

    let point = arr2(&[[0.1, 0.2, 0.3]]);
    let result = evaluate_sample(
        &mut loaded.state,
        &EvaluateSamples {
            process_id: Some(0),
            integrand_name: Some("one_loop".into()),
            use_arb_prec: false,
            minimal_output: false,
            return_generated_events: None,
            momentum_space: false,
            points: point.view(),
            integrator_weights: None,
            discrete_dims: None,
            graph_names: None,
            orientations: None,
        },
    )?;

    println!("{result}");
    std::fs::remove_dir_all(state_folder)?;
    Ok(())
}
```

Compile and run with GammaLoop's public Symbolica build selector:

```
SYMBOLICA_OEM_LICENSE=SYMBOLICA_OEM_GAMMALOOP cargo run
```

The facade owns the supported embedding boundary
Prefer `StateLoadOption`, `CliSession`, and the typed request objects from `gammaloop-api`. `gammaloop` exposes lower-level implementation types, but their presence in Rustdoc is not a stability promise. The interface guide maps the supported workflow to the complete Rust reference.

The docs harness compiles this source against the current workspace. Running it additionally requires the native toolchain and a Symbolica license mode appropriate for the user.

6 Physics scope and Local Unitarity

GammaLoop is alpha-stage research software for perturbative quantum-field-theory calculations. Its present, demonstrable calculation path is partonic: it constructs amplitude integrands and uses Local Unitarity forward-scattering representations for differential cross sections, then evaluates or integrates selected contributions. It is not currently a general hadron-level event generator or a promise that every process allowed by a model is numerically supported.

> **Intended reader**
> This chapter is for physicists deciding whether GammaLoop is appropriate for a calculation. The command-line workflow requires no Rust knowledge. Read the kinematics and normalization contract before interpreting a value, and treat compiled loop-count limits as software bounds rather than demonstrated perturbative reach.

6.1 What Local Unitarity changes

At fixed perturbative order, a conventional calculation separates real-emission and virtual contributions even though their infrared singularities cancel only in the sum. The Local Unitarity representation uses Loop–Tree Duality for each forward-scattering graph to express those degrees of freedom on common integration variables. Contributions that approach the same soft or collinear configuration can then cancel locally, before a Monte Carlo integral relies on cancellations between separately integrated numbers.

The defining Local Unitarity paper proves this representation for processes without initial-state collinear singularities and introduces local ultraviolet counterterms. The raised-propagator and local-renormalization paper extends the formalism with distributional cutting rules and a local R-operation treatment. Those are statements about the method. A method valid at arbitrary perturbative order is not evidence that this release implements every order, process, initial state, or renormalization choice.

For one Monte Carlo point, GammaLoop can produce several cut or counterterm contributions in one or more correlated event groups. The observable layer commits the complete group list as one statistical sample. Treating the members as independent events would discard the local-cancellation relation and underestimate correlations.

6.2 Capability envelope in this release

The labels below are deliberately conservative:

- **Supported** means an explicit implementation path was identified; it does not certify every process or numerical regime.
- **Conditional or experimental** means feature-, backend-, model-, or workflow-dependent.
- **Unsupported** means the accepted value reaches a placeholder, explicit error, or panic.
- **Unverified** means the source does not justify a public scientific guarantee.

Area	Classification	Current boundary
Calculation objects	Supported, scoped	Amplitude integrands and forward-scattering cross-section integrands are generated. Numerical support must still be established for the selected topology and contribution.
Initial states and PDFs	Partonic only	The current external kinematics are fixed momenta and helicities, with a PDF factor of one. There is no PDF convolution or factorization scale. The built-in flux path supports exactly one or two incoming particles.
Orders and selectors	Conditional filters	`{n}` constrains an amplitude graph's loops or the sum on two cut sides, while `{{n}}` constrains forward-graph loops. Outside

Area	Classification	Current boundary
		the bracketed block, unpowered constraints populate the amplitude filter and powered constraints populate the cross-section filter, whose exponent is currently discarded. Bracketed named orders bound model-resolved additional cut content on the cross-section path. These mechanisms are not interchangeable and do not provide a universal LO/NLO/NNLO mapping.
Models	Conditional	Built-in Standard Model, scalar, and scalar-gravity JSON models are available; other JSON models can be imported. UFO import requires the optional Python-backed feature and its loader.
External polarizations	Scoped	Automatic sums are implemented for scalar, spinor, and vector states, with Feynman or light-like axial gauge for vectors. Higher-spin polarization replacement is unsupported.
Masses and widths	Real masses only	Real masses enter evaluation. Complex masses are unsupported. Width names are model metadata, but no finite-width or complex-mass scheme was verified.
Subtraction and renormalization	Conditional or experimental	Local UV settings and several vacuum-integral backends exist. Threshold subtraction is experimental, and the physical meaning of every exposed prescription still needs a method-author review.
Integration components	Scoped	Real and imaginary components can be integrated separately. The accepted `both` setting is not implemented.
Observables	Supported; requires user validation	Particle and jet quantities, selectors, histograms, and kT-family clustering are available. GammaLoop does not establish the infrared and collinear safety of an arbitrary user-defined observable.
Licenses and tools	Conditional	The Rust crates use MIT or Apache-2.0 terms, while Symbolica has separate license conditions. FORM, vacuum-integral backends, UFO import, and drawing tools have independent availability requirements.

6.3 Decide whether a calculation is in scope

Before generating graphs, write down these physics choices outside the run card:

1. Is the requested quantity partonic with fixed external kinematics, or does it require PDFs, a factorization scale, beam structure, or more than two incoming particles?
2. Is it an amplitude or a differential cross section, and which exact graph, loop, cut, and coupling-order filters define the requested contribution?
3. Does the active model use only supported external spins and real masses? If a width is physically essential, do not infer a finite-width scheme from imported metadata.
4. Which spin and color sums or averages, projectors, flux, symmetry factors, and units belong in the normalization? Record them using the conventions checklist.
5. Is the observable infrared and collinear safe for the selected real/virtual combination? This is a physics precondition, not an automatic validation step.
6. What maintained limit, independent implementation, or published number will be used to validate the result and its Monte Carlo uncertainty?

Method scope versus software evidence
The first-state tutorial proves that a model, generated integrand, settings, and persisted state fit together; it does not validate a physical cross section. Use the partonic cross-section regression card and the scalar-

topology targets as concrete implementation sources to inspect. The former still uses a powered selector whose exponent is not enforced, so it is not an audited perturbative-order or normalization exemplar. Publish a number only after recording its contribution definition, normalization, convergence, and an independent comparison.

7 Kinematics, normalization, and weights

This page assembles the source-verifiable scientific conventions needed to interpret a GammaLoop sample. It intentionally separates factors supplied by the runtime from factors that remain the physicist's responsibility. Where the implementation does not establish a universal convention, the text says so instead of filling the gap from customary notation.

7.1 Four-vectors and momentum flow

External four-vectors use component order (E, p_x, p_y, p_z) and metric signature $g = \mathrm{diag}(+1, -1, -1, -1)$, so

$$p \cdot q = E_p E_q - (p_x q_x + p_y q_y + p_z q_z), \quad p^2 = E^2 - p_x^2 - p_y^2 - p_z^2.$$

Stored external momenta are physical, normally positive-energy vectors. A separate process signature marks a particle as initial or final; GammaLoop does not require the user-facing list to store every final momentum with a minus sign. When one momentum is dependent, it is reconstructed to satisfy

$$\sum_i^{\text{initial}} p_i - \sum_j^{\text{final}} p_j = 0.$$

The runtime field `e_cm` is a construction and normalization scale. If it is omitted for fixed momenta, the current fallback is the mean absolute value of their nonzero components, not a physical reconstruction of $\sqrt{s}$. Supply `e_cm` explicitly whenever its physical meaning matters.

7.2 Helicity and polarization

Fixed external helicities use `-1`, `0`, or `+1`. Zero is the physical and documented convention for a scalar card, but the current evaluator ignores rather than enforces a scalar's helicity. The automatic external-momentum generator currently assigns `+1` to every external entry; that is a fixed-polarization default, not an unpolarized calculation.

Use `summed` for a polarization sum and `summed_averaged` for the same replacement with the implemented spin-state average. The setting acts on every external entry explicitly marked that way; incoming particles are not averaged merely because they are incoming. Implemented averaging factors are 1 for a scalar, $\frac{1}{2}$ for a spinor, $\frac{1}{2}$ for a massless vector, and $\frac{1}{3}$ for a massive vector. Automatic higher-spin polarization sums are unsupported.

> **Color is a separate normalization decision**
> No general automatic incoming-color average was identified. A maintained calculation may supply a color projector or global prefactor explicitly. Record that choice rather than assuming that `summed_averaged` includes color.

7.3 Flux and reporting units

With the flux factor enabled, the implemented one-incoming-particle factor is

$$F_{1\text{ in}} = \frac{1}{2E},$$

and the implemented two-incoming-particle factor is

$$F_{2\,\mathrm{in}} = \frac{1}{4\sqrt{(p_1 \cdot p_2)^2 - m_1^2 m_2^2}}.$$

Zero or three-or-more incoming particles are not implemented by this default path. Reporting units resolve to `pb`, `fb`, `ab`, `mb`, or `none`; `auto` selects picobarns for more than one incoming particle and no conversion otherwise. The fixed factor $3.89379372171859 \times 10^8$ relative to picobarns is applied only by the two-incoming-particle flux branch. The one-incoming-particle branch returns $F_{1\,\mathrm{in}}$ without a reporting-unit conversion. Disabling the flux factor also bypasses the conversion branch.

The code describes input scales as model energy units, while the fixed picobarn conversion is the usual natural-unit form associated with GeV. Until the energy-unit premise is made an explicit model contract, do not claim that arbitrary model units convert correctly to pb.

7.4 From an integrand value to a Monte Carlo sample

For the public sample-evaluation contract, the exact weighting relation is

$$w_{\mathrm{sample}} = I_{\mathrm{returned}} J_{\mathrm{parameterization}} w_{\mathrm{MC}}.$$

`integrand_result` is the returned integrand before the parameterization Jacobian, and `integrator_weight` excludes that Jacobian. During integration, GammaLoop first multiplies by the parameterization Jacobian and then applies the Monte Carlo sample weight. For a cross section, the returned integrand already contains the enabled flux and reporting-unit factor.

An event group may retain an original contribution, threshold counterterms, and one full multiplicative factor. Its implemented reconstruction is

$$w_{\mathrm{event}} = \left(w_{\mathrm{Original}} + \sum_i w_{\mathrm{CT,i}} \right) w_{\mathrm{FullMultiplicativeFactor}}.$$

All groups originating from one Monte Carlo point are combined before one statistical sample is committed to a histogram bin. They are correlated contributions, not independent events. The events and observables guide shows the corresponding Python and Rust records.

7.5 Normalization ownership checklist

Factor	Usual owner here	What to record
Polarization sum and spin average	Run card	The explicit helicity mode for every external particle and the vector gauge choice.
Color sum or average	Physicist / projector	The projector or prefactor and its normalization; do not infer a universal automatic average.
Graph multiplicity and signs	Generated graph factor	Automorphism, fermion-loop, external-ordering, and grouping contributions visible in the generated state.
Flux and output units	Runtime when enabled	Incoming momenta and masses, the selected unit, and whether the flux branch was disabled.
Parameterization Jacobian	Integration driver	The parameterization and whether a reported value is before or after its Jacobian.
Monte Carlo weight	Integrator	The sample weight, seed, component, and correlation boundary.
Subtraction contributions	Correlated event group	Original and counterterm weights, signs, and the common multiplicative factor.
Overall physical normalization	Calculation definition	Any remaining coupling, projector, color, symmetry, phase-space, or conventional factor used for comparison.

Generated graph factors retain several combinatorial and fermionic contributions, but the source audit did not establish one universal statement that these plus a chosen projector form the complete physical symmetry and averaging factor for every process. Inspect the generated factor and write the full normalization used by the comparison calculation.

7.6 Scales and scheme names

Keep these settings distinct:

- `mu_r` is the renormalization scale and enters the parameter set as its square;
- `m_uv` is a UV reference mass;
- `renormalization_localization_scale` controls localization of local UV counterterms;
- MUV, PolePart, OS, IR, Unsubtracted, and VaccuumLimit are accepted local-counterterm prescription names whose full physical definitions still require a method-author contract; OS, IR, and VaccuumLimit reach explicit unimplemented paths in at least one current counterterm evaluator; and
- Vakint's default `MSbar` choice describes its vacuum-integral normalization and must not be used as a synonym for GammaLoop's MUV prescription.

No factorization scale belongs to the current fixed-partonic external-kinematics path.

7.7 Terms that are easy to overload

- A **GammaLoop state** is the persisted calculation workspace, not a quantum state and not an integration checkpoint.
- An **integration workspace** owns resumable Monte Carlo iterations for selected `(process, integrand)` slots; its summary JSON is not the authoritative checkpoint.
- An **integration coordinate** is a parameterization coordinate unless `momentum_space` is explicitly selected, in which case each loop momentum is a flattened `(p_x, p_y, p_z)` triplet.
- A **target** is a comparison value used to report a delta. It is not the relative or absolute accuracy that stops integration.
- An **event group** contains correlated contributions from one Monte Carlo point. A **graph group** is a generation-time grouping of related graphs and multiplicity/numerator factors; the two are not interchangeable.

> **Conventions not yet established as universal contracts**
> Before publishing a result, obtain an explicit calculation-level statement for all 2π, i, sign, contour, and phase-space-measure factors; overall amplitude and cross-section normalization; color sums and averages; remaining symmetry factors; powered coupling constraints; the LO/NLO meaning of graph filters; finite-width treatment; and the selected renormalization prescription. The current source does not justify silently supplying any of these from convention alone.

The e+ e- partonic regression card is useful for tracing implemented factors, but it is not yet a normalization exemplar: its powered selector exponent is not enforced. For any audit, record the model, contribution filters, helicities, projectors, runtime settings, observable definition, and independently reviewed perturbative-order definition.

8 Tutorial

This tutorial builds the GammaLoop command-line interface and uses the included one-loop `g g -> h h h` run card to create a state that can be inspected and resumed. Compilation and process generation can take longer than a `--help` smoke test.

For the shortest installation and first-run path, begin with Using GammaLoop from the command line. Then return here to create, inspect, and resume the larger bundled example.

What this example establishes
The card selects one fixed-helicity, one-loop amplitude contribution and supplies an explicit `1/8` color projector. It demonstrates software state creation, inspection, and resumption; it is not an unpolarized, automatically color-averaged cross section. Read the method and capability boundary and normalization conventions before interpreting either integration block as a physical result.

8.1 Prerequisites

Work from a GammaLoop source checkout. The supported development environment is the repository Nix shell; without Nix you need Rust 1.85 or newer, `just`, a recent GNU toolchain, and the dependencies described in the root README. The bundled Standard Model used below does not need the UFO loader. A Symbolica license may be required by your build and use case.

```
nix develop
just build-cli
./gammaloop --help
```

The last command should print the one-shot CLI options and command tree. The `./gammaloop` wrapper selects the binary built under `target/dev-optim/`.

Verification scope and cost
The docs harness checks these shell blocks for syntax; it does not build or execute GammaLoop. Verify a clean checkout manually in `nix develop`. A cold CLI build and process generation are the high-resource path and can take tens of minutes, while reopening a valid state is normally much cheaper. The invariant for this tutorial is a resumable state with the files listed below, not a completed Monte Carlo integration.

8.2 Generate an example process

The run card declares a state folder, settings, and named command blocks. Run its `generate` block and persist the result on exit:

```
./gammaloop --clean-state \
  ./examples/cli/gg_hhh/1L/gg_hhh_1L.toml \
  run generate -c "quit -o"
```

`--clean-state` removes the resolved state first
Use it only for this first, reproducible run. Omit it when resuming work you want to keep. Its generated argument reference records the exact valueless-flag semantics. The example card resolves its state to `examples/cli/gg_hhh/1L/state` and requests ten worker cores; make a private copy of the card before changing either value.

The block imports `sm-default.json`, generates the selected one-loop pentagon amplitude contribution, applies the card's explicit helicity and color choices, builds its integrand, saves DOT data, and writes the state. A successful run exits without an error and leaves at least `run.toml`, `global_settings.toml`, `default_runtime_settings.toml`, and `processes/` in the state folder. `run.toml` records the commands and settings needed to replay the run; it is not merely a log file.

8.3 Resume the saved state

Load the saved state and ask the active session to display its processes:

```
./gammaloop -s ./examples/cli/gg_hhh/1L/state \
  run -c "display processes; quit -o"
```

Success means the display includes the generated `gg_hhh` process and the command exits while keeping the same state. You can now run the card's `integrate_euclidean` or `integrate_physical` block, but those are deliberately beyond the first-success path because they request a substantial Monte Carlo integration and require a separate normalization and validation plan.

8.4 Inspect before expensive work

Reopen the state read-only and inspect both the generated structure and the effective integrator settings before launching either integration block:

```
./gammaloop --read-only-state \
  -s ./examples/cli/gg_hhh/1L/state \
  run -c "display integrand -p gg_hhh -i 1L; display settings process -p gg_hhh -i 1L
integrator; quit"
```

`display integrand` reports the generation backend, compiled record size, graph and graph-group counts, and detailed orientation, loop-momentum-basis, cut, and threshold tables. The settings command resolves the values attached to this generated integrand, which may differ from the current defaults. Check these records together: an unexpected graph count points back to process generation, while an unexpected sample budget, target accuracy, or seed belongs to runtime settings.

> **Read-only inspection is not persistence**
> `--read-only-state` prevents commands from writing inside the active state directory and disables file logging there. It does not make the in-memory session immutable. This command intentionally exits without `-o`; change settings in a separate writable session or in the run card that reconstructs the state.

The exact fields and selectors are kept current in the generated integrand display and process-settings references.

8.5 Troubleshooting and next steps

- If the wrapper cannot find a binary, rerun `just build-cli` and confirm that `target/dev-optim/gammaloop` exists.
- If compilation or process generation exhausts local resources, copy the card and reduce the values under `cli_settings.global.n_cores`; do not edit a state midway through generation.
- If startup reports a state fingerprint or settings mismatch, replay the card with a new state folder instead of transplanting only its `processes/` directory.
- Use `./gammaloop help generate` to inspect the flags supported by your installed version, then continue with the process-generation guide and the generate-command reference.

> **Example run card**
> See the complete gg_hhh run card before adapting the process, state location, or resource settings for your own calculation.

9 Process generation and state workflows

GammaLoop describes a calculation with a TOML run card and stores its reusable state in a directory. Start the command-line interface through the repository wrapper (`./gammaloop`) or the Cargo-built executable, then generate processes, integrands, and integration jobs within that state.

9.1 Choose a generation mode

The interactive command tree separates cross sections, amplitudes, and operations on processes already stored in the active state:

```
generate xs INITIAL > FINAL ...
generate amp INITIAL > FINAL ...
generate existing ...
generate help xs
```

`xs` constructs forward-scattering graphs and filters compatible Cutkosky cuts. `amp` constructs ordinary amplitudes. Their final-state lists therefore have different meanings even when the printed particle syntax is identical. Use `generate help <mode>` inside the stateful CLI to see the flags supported by your installed version.

9.2 Process specification grammar

Process specifications accept `to` or `>` between space-separated initial and final states. Braces express alternatives in a cross-section final state, while `{}` denotes an empty state:

```
e+ e- > mu+ mu-
e+ e- > { Z Z, a a, H H }
{} > {}
```

The empty-initial-state form is generation grammar, not evidence that the default cross-section flux evaluator supports a zero-incoming-particle calculation. Its implemented flux branches require exactly one or two incoming particles.

A slash vetoes particles and a vertical bar selects an allow-list. The words `veto` and `only` are equivalent forms. Outside the bracketed perturbative block, coupling constraints use the active model's coupling-order name:

```
e+ e- > d d~ g / u c QED==2 QCD>=2 QCD<=4
e+ e- > mu+ mu- | g ghG QED==2 QCD>=2
```

> **Powered and unpowered constraints are not interchangeable**
> An unpowered spelling such as `QCD==2` builds an amplitude-side coupling-order filter. A powered spelling such as `QCD^2==2` builds a cross-section-side coupling-order filter, but the parsed exponent is currently discarded. Substituting one spelling for the other changes which filter container is used and still does not define a squared amplitude or a complete cross-section order. Production work therefore needs a scientifically reviewed selector set; do not infer one from the spelling alone.

The bracketed perturbative block is intentionally distinct from ordinary flags. For `amp`, `{n}` fixes the amplitude graph's loop count; for `xs`, it fixes the sum of loop counts on the two cut sides. `{{n}}` fixes the forward graph's loop count. On a cross-section process, bracketed `QCD=n` or the shorthand `QCD` resolves the model's unresolved-particle set and permits at most the summed requested count of additional cut particles from those sets; it is not an exact graph coupling power. No corresponding amplitude-side selection effect has yet been established for that bracketed named-order spelling:

```
e+ e- > Z [ {1} {{2}} QCD=2 QED=1 ]
e+ e- > Z [ QCD ]
```

These selectors are generation filters, not universal definitions of LO, NLO, or NNLO. Record the amplitude-side loops, forward loops, unresolved-cut allowance, and coupling filters explicitly for the chosen generation mode; use the physics-scope guide to distinguish that calculation definition from the formal perturbative reach of Local Unitarity.

> **Resolve names through the active model**
> Particle names, antiparticles, coupling-order names, and allowed interactions come from the model loaded in the current state. A syntactically valid specification can still be rejected when that model does not define a requested name.

9.3 Filters, grouping, and multiplicity

Generation flags control topology filters, loop ranges, cut blobs and spectators, symmetrization, and numerator-aware grouping. These choices are physics-significant. In particular, grouping can combine graph multiplicities and numerator ratios; external-fermion symmetrization requires the caller to retain the associated ordering signs. Run `generate help <mode>` before choosing filters to see the available aliases, defaults, values, and argument counts.

The generated graph's overall factor keeps separate contributions for automorphisms, internal fermion loops, external-fermion ordering, numerator-independent symmetry grouping, and numerator-dependent grouping. It does not establish a universal incoming-color average. Persist the state and inspect that factor together with any explicit projector when auditing diagram normalization instead of replacing it with an assumed factorial; the normalization checklist assigns each factor to its owner.

9.4 A maintainable generation card

The following reduced card uses the repository's scalar model and the same one-loop bubble command exercised by the maintained scalar-topology example. Save it as `manual-bubble.toml` and run `./gammaloop --clean-state manual-bubble.toml run generate -c "quit -o"` from the repository root.

```toml
commands = []

[cli_settings.state]
folder = "./gammaloop_state/manual-bubble"
name = "manual_bubble"

[[command_blocks]]
name = "generate"
commands = [
  "import model scalars-default.json",
  "generate amp scalar_1 > scalar_1 [{1}] --allowed-vertex-interactions V_3_SCALAR_122 -p
bubble -i scalar_bubble",
  "generate",
  "save state -o",
]
```

The acceptance invariant is a persisted state in which `display processes` identifies process `bubble` and integrand `scalar_bubble`; replaying the card with a clean destination must produce the same named objects. The exact command and setting semantics are linked from the amplitude-generation reference, state-persistence reference, and state-folder setting. The full maintained source is the scalar bubble run card.

> **Interpret the first failing stage**
> An unknown particle or `V_3_SCALAR_122` error means the scalar model was not imported or no longer exposes that interaction. A process with no retained diagram points to the process specification or generation filters. A generated process that disappears after restart means `save state -o` did not complete or the resumed `-s` path differs from `cli_settings.state.folder`.

9.5 From generation to evaluation

A reproducible run keeps its stages explicit:

- import or select the model before resolving a process specification;
- generate the process and inspect the retained diagrams;
- generate the requested integrand representation;
- set kinematics, sampling, stability, and subtraction settings;
- evaluate samples or run an integration command block;
- exit with persistence enabled when the state should be resumed.

9.6 Select integration slots and targets

An integration slot is one `(process, integrand)` pair. Repeat `-p/--process` and `-i/--integrand-name` in matching order to integrate several slots together. By default those slots train and sample a shared grid, so their sample shapes must be compatible; add `--uncorrelated` when each slot needs an independent grid.

A single `--target re im` (or `--target re,im`) applies the same known comparison value to every selected slot. Multi-slot runs can instead use one keyed value per slot:

```
integrate \
  -p bubble_no_integrated_UV -i scalar_bubble_below_thres \
  -p bubble -i scalar_bubble_below_thres \
  --target bubble_no_integrated_UV@scalar_bubble_below_thres=1.471664021721597e-2,0.0 \
           bubble@scalar_bubble_below_thres=2.7029875684552542e-3,0.0 \
  --workspace-path ./integration/bubble-below-threshold \
  --n-cores 1
```

Targets annotate result deltas; they do not replace `integrator.target_relative_accuracy`, `integrator.target_absolute_accuracy`, or the sample-budget settings that control convergence. The maintained scalar bubble run card contains the complete settings and both below- and above-threshold target sets. The generated integration reference is authoritative for selectors, renderers, batching controls, and allowed values.

9.7 Resume or replace an integration workspace

An integration workspace is a completed-iteration checkpoint, not a second GammaLoop state. Without `--workspace-path`, writable sessions use `STATE_FOLDER/integration_workspace`; read-only sessions use `./integration_workspace_STATE_NAME` (or `./integration_workspace`) so they do not write into the active state.

- Without `--restart`, GammaLoop resumes when the workspace already contains a compatible checkpoint. Slots, shared-versus-uncorrelated sampling, effective model parameters, and generated-integrand fingerprints must match. Saved targets and non-model runtime settings win over changed values and produce a warning.
- With `--restart`, GammaLoop removes that specific workspace and starts the integration from scratch. Use it when changing slots, sampling correlation, generated integrands, or settings that should take effect. It does not mean "resume".
- `--show-summary-only` reads the last completed result without evaluating more samples and cannot be combined with `--restart`.

The workspace root contains `manifest.json`, the latest user-facing `integration_result.json`, per-slot settings and results under `integrands/`, and the internal resume checkpoint under `state/integration_state.bin`. Observable resume snapshots live at `state/observables/<process@integrand>/latest.json`. Configured user-facing output is written per slot as `integrands/<process@integrand>/observables_final.json` or `integrands/<process@integrand>/observables_final.hwu`. `--write-results-for-each-iteration` additionally retains numbered result and observable history at the corresponding root or slot boundary.

> **Archive the right boundary**
> `integration_result.json` is a presentation snapshot, not the authoritative resume state. Keep the complete integration workspace when a run must be resumed, and keep the GammaLoop state/run card that produced the referenced integrands. A copied result JSON alone is suitable for analysis, not continuation.

Keep the complete state together
`run.toml` records boot settings, reusable command blocks, and top-level commands. Generated process and integrand artifacts live beside it. Resume with `-s STATE_FOLDER`; replay the run card to reconstruct the state from scratch. Copy or archive the entire state directory rather than moving a generated subdirectory on its own, because settings and fingerprints are checked together.

Contributor-facing ownership and data flow are documented in the GammaLoop architecture and the UV renormalization architecture.

10 Inspect events and observables

GammaLoop can evaluate selected points without launching a full adaptive integration. This is the shortest path for inspecting the generated cut events, testing selectors, and checking that histograms receive the expected weights. It is an investigative workflow: a few chosen points cannot establish convergence or replace the persisted result of an integration.

10.1 Start from the differential regression example

The repository contains matching Python and Rust journeys for the differential Local Unitarity process `e+ e- > d d~ g`. Their run cards generate the real two-graph process and configure:

- event generation and additional event weights;
- jet transverse-momentum, jet-count, and down-quark-energy quantities;
- a leading-jet selector; and
- continuous and discrete histogram observables.

What this example verifies
This is an API and event-pipeline regression fixture, not a scientifically reviewed perturbative-order or normalization benchmark. Its run card uses a powered coupling selector, but the current filter construction discards that exponent. Keep it for inspecting events and observable plumbing; define and validate production contributions independently.

The run card's `display_named_settings_examples` block lets you inspect each named object before evaluation:

```
display quantities -p #0
display quantities -p #0 leading_jet_pt
display selectors -p #0 leading_jet_pt_cut
display observables -p #0 leading_jet_pt_hist
```

Use these displays as a preflight check. A missing quantity name belongs to configuration; an empty event list belongs to event generation or selection; an unexpected histogram belongs to the quantity, phase, binning, or weights. Keeping those stages separate is more informative than debugging the final number alone.

10.2 Run the Python journey

Build the native package, then run the maintained script from the repository root. It creates a fresh state beside the script, generates the configured process, evaluates one integration-space point, evaluates a two-point batch, and evaluates one momentum-space point:

```
nix develop
uv venv .venv
source .venv/bin/activate
just build-api
python examples/api/python/epem_a_ddxg_xs_LO/inspect_events.py
```

The exact workflow is kept in the Python event-inspection script and its differential run card. The core API sequence is:

```python
from pathlib import Path

import numpy as np
from gammaloop import GammaLoopAPI

example = Path("examples/api/python/epem_a_ddxg_xs_LO")
api = GammaLoopAPI(
    state_folder=example / "state",
    boot_commands_path=example / "run.toml",
    clean_state=True,
)
api.run("run display_named_settings_examples")

info = api.get_integrand_info()
assert info.kind == "cross section"
assert info.graph_count == 2
assert info.graph_group_count == len(info.graph_groups)
assert all(
    sum(graph.is_master for graph in group.graphs) == 1
    for group in info.graph_groups
)

point = [0.17, 0.31, 0.53, 0.23, 0.41, 0.67]
single = api.evaluate_sample(point, return_events=True)
assert single.parameterization_jacobian is not None
assert single.stability_results
assert single.event_groups

points = np.array([
    point,
    [0.11, 0.29, 0.47, 0.19, 0.37, 0.59],
], dtype=float)
batch = api.evaluate_samples(points, return_events=True)
assert len(batch.samples) == 2
assert all(sample.event_groups for sample in batch.samples)
assert len(batch.observables["leading_jet_pt_hist"].bins) == 8
assert len(batch.observables["jet_count_hist"].bins) == 6
assert batch.observables["leading_jet_pt_hist"].sample_count == 2
assert batch.observables["jet_count_hist"].sample_count == 2

print(single.integrand_result, single.generated_event_count)
for group in single.event_groups:
    for event in group.events:
        print(event.cut_info.graph_id, event.weight, event.outgoing_momenta)
print(batch.observables["leading_jet_pt_hist"].sample_count)
```

`return_events=True` temporarily requests returned events for that call; it does not permanently rewrite the integrand setting. Every `EventGroup` contains correlated accepted events produced by one graph group. Each event carries incoming/outgoing momenta, its primary complex weight, cut and graph identifiers, and any configured additional weights.

The normalization and weight contract defines which factors are already present in an event weight and why all contributions from one Monte Carlo point must be committed as one statistical sample.

> **Single and batch results have different aggregation boundaries**
> `evaluate_sample` returns one `EvaluationResult`: its event groups and observable snapshot be-

> long to that sample. `evaluate_samples` returns per-row `SampleEvaluationResult` objects and one `BatchEvaluationResult.observables` dictionary merged across the whole batch. Do not read a batch histogram as if it belonged to the last row, and do not add the same batch snapshot once per sample.

The histogram values are immutable snapshots. Inspect `bins`, underflow/overflow, raw sums of weights and squared weights, entry counts, and the aggregate statistics before deriving a plot. Keep raw statistics when combining results; rounded rendered values are not a merge format.

10.3 Run the Rust journey

The Rust example loads the same kind of run card through `StateLoadOption`, executes the display block through a CLI session, then calls the typed evaluation functions. It is a `rust-script` program so its dependency revision is recorded with the example:

```
NO_SYMBOLICA_OEM_LICENSE=1 \
EXTRA_MACOS_LIBS_FOR_GNU_GCC=T \
SYMBOLICA_LICENSE="$SYMBOLICA_LICENSE" \
rust-script --debug examples/api/rust/epem_a_ddxg_xs_LO/inspect_events.rs
```

The documentation gate compiles the request boundary below against the workspace API. It keeps the event-return override and every required field synchronized even though the licensed process evaluation remains outside the pull-request tier:

```
use gammaloop_api::commands::evaluate_samples::EvaluateSamples;
use ndarray::arr2;

let points = arr2(&[[0.17, 0.31, 0.53, 0.23, 0.41, 0.67]]);
let request = EvaluateSamples {
    process_id: None,
    integrand_name: None,
    use_arb_prec: false,
    minimal_output: false,
    return_generated_events: Some(true),
    momentum_space: false,
    points: points.view(),
    integrator_weights: None,
    discrete_dims: None,
    graph_names: None,
    orientations: None,
};
assert_eq!(request.return_generated_events, Some(true));

let loop_momenta = arr2(&[[
    0.11, -0.07, 0.19, // k1 = (px, py, pz)
    -0.13, 0.05, 0.29, // k2 = (px, py, pz)
]]);
let momentum_request = EvaluateSamples {
    process_id: None,
    integrand_name: None,
    use_arb_prec: true,
    minimal_output: true,
    return_generated_events: None,
    momentum_space: true,
    points: loop_momenta.view(),
    integrator_weights: None,
    discrete_dims: None,
    graph_names: None,
    orientations: None,
```

```
};
assert_eq!(momentum_request.points.ncols(), 2 * 3);
```

Read the Rust event-inspection script with its run card. The ordinary `EvaluateSamples` path returns the cross-language `f64` boundary. Rust callers that must retain the active numeric precision use `EvaluateSamplesPrecise`; arbitrary-precision internal evaluation does not make the ordinary result fields arbitrary-precision. `use_arb_prec=true` forces arbitrary-precision (Arb) internal evaluation, using the configured Arb stability level when available and a default Arb level otherwise. The sample evaluation contract defines this flattened momentum layout and the precision behavior once for CLI, Rust, and Python.

Verification and resource cost
The pull-request gate parses the commands, compiles the Python source, and type-checks the Rust request literal, but does not run process generation or native evaluation. The maintained scripts are the real acceptance path and require a built API, the repository's scientific dependencies, and an appropriate Symbolica license. They create or replace only their example-local `state` directory because they opt into `clean_state=True`; remove that option before pointing at work you need to keep.

10.4 From point inspection to integration output

Selected points answer structural questions: whether events are produced, how graph/cut metadata is grouped, which selector accepts them, and which histogram bins are filled. A production result still needs an integration workspace, target-accuracy settings, iteration diagnostics, and the final observable output. Continue with the state and integration-workspace guide for resume semantics and the distinction between `integration_result.json`, `observables_final.*`, and the authoritative checkpoint.

Contributors changing progress reporting, cancellation, or terminal ownership should also read the integration dashboard architecture.

The exact Python return types and fields are linked from the interface guide and generated Python API.

11 CLI, Rust, and Python APIs

11.1 Rust packages

The Rust API spans two primary packages.

- `gammalooprs` contains the physics implementation, data model, integrands, integration, observables, and lower-level algorithms.
- `gammaloop-api` provides the supported state-loading API, session/command integration, CLI assembly, and the PyO3 extension used by the Python distribution.

Load a state with `StateLoadOption::load()`. Its options select a state directory, boot card, logging overrides, read-only behavior, and optional settings overrides. Once loaded, callers can create a CLI-style session or use dedicated structured operations. A raw command string is the compatibility fallback, not a replacement for a typed operation where one exists.

```
use std::path::PathBuf;
use gammaloop_api::{state::CommandHistory, StateLoadOption};

let mut loaded = StateLoadOption {
    state_folder: Some(PathBuf::from("./state")),
    ..StateLoadOption::default()
}.load()?;
```

```rust
let command = CommandHistory::from_raw_string("display settings global")?;
loaded.cli_session().execute_command(command)?;
```

Some Rust types are public so that GammaLoop's packages can work together, but are not intended as stable integration points. Prefer the state-loading and structured-operation APIs described here; use the Rust orientation to choose a crate, then use that revision's Rustdoc for exact signatures, trait implementations, and lower-level types.

11.1.1 Rust reference map

This authored map separates the supported workflow from the much larger compiled surface. Follow these exact Rustdoc paths instead of beginning in an arbitrary internal module:

- **Load and command a state:** begin with `StateLoadOption`, retain the returned `LoadedState`, and execute parsed `CommandHistory` values through `CliSession`.
- **Replay and inspect:** use `RunHistory` for run cards and `IntegrandInfo` for structured graph-group, orientation, loop-basis, cut, and threshold metadata.
- **Evaluate samples:** construct `EvaluateSamples` for the ordinary `f64` boundary. Use `EvaluateSamplesPrecise` only when the caller must retain the active `f64`, `f128`, or arbitrary-precision result type.
- **Work below the facade:** the `gammalooprs` Rustdoc includes the integrand contract, runtime settings, precision-specific result records, and the mergeable histogram snapshot boundary. Treat public items not named by a maintained workflow as implementation-facing unless their crate documents a stronger compatibility promise.

11.2 Python packaging

A standalone GammaLoop distribution
The Python distribution and import package are both named `gammaloop`. Its compiled backend is `gammaloop._gammaloop`, which applications normally access through the public package. This is separate from the `symbolica.community.*` modules used by Spenso, Idenso, and Vakint; installing GammaLoop does not install those community modules as independent packages.

The main Python entry point is `GammaLoopAPI`. Its constructor loads or creates one stateful session:

```python
from gammaloop import GammaLoopAPI

gl = GammaLoopAPI(
    state_folder="./state",
    boot_commands_path="./run.toml",
)
gl.run("display settings global")
```

The Python package requires Python 3.11 or newer. Run `just build-api` when building the bindings from a source checkout.

11.2.1 Python reference map

The generated module reference covers every registered public export, but the objects form a few connected workflows rather than forty unrelated entry points:

- **Session lifecycle:** construct `GammaLoopAPI`, then use `run`, the history getters, and `SettingsValue` to automate the same state and commands as the CLI.
- **Point evaluation:** `evaluate_sample` returns `EvaluationResult`, whose `sample` is a `SampleEvaluationResult`. `evaluate_samples` returns `BatchEvaluationResult` with the same sample records and one batch-level observable snapshot.

- **Generated-integrand structure:** start at `IntegrandInfo`, then follow its graph groups into graph, orientation, loop-momentum-basis, cut, and threshold records. These objects describe compiled structure; they do not mutate it.
- **Events and observables:** evaluation records lead to `EventGroup` and `Event`. For caller-owned aggregation, `HistogramAccumulator` produces immutable `HistogramSnapshot` records with raw statistics that remain mergeable and reconstructible.

Full integrations still use the command interface
The current Python module does not expose a supported structured `integrate()` method. Run an integration through `GammaLoopAPI.run("integrate ...")` or the CLI and consume its persisted workspace/results. Several integration-result record classes are registered by the native module for ongoing API work, but without a public producer they are not part of the curated Python boundary yet.

Result ownership
Objects returned by evaluation and integration are snapshots. Mutating a Python list or dictionary derived from them does not update the live GammaLoop session. The explicit mutable exception is `HistogramAccumulator`, whose methods update caller-owned aggregation state.

Caller-owned histogram correlation limit
Do not yet use `HistogramAccumulator.fill_continuous_sample` or `fill_discrete_sample` to replay raw correlated event entries when more than one entry can land in the same bin. The helper counts the call as one sample but currently accumulates the entries' squared weights separately, unlike the native observable path, which groups their weights before recording one bin sample. Keep production histograms in the configured native observable pipeline until this statistical contract is aligned.

11.2.2 Aggregate independent histogram samples

This safe standalone use keeps one entry per bin in each independent sample. Merge pending worker state before committing it, then retain raw sums and squared sums for later combinations:

```
from gammaloop import HistogramAccumulator

left = HistogramAccumulator.continuous("energy", 0.0, 4.0, 4)
right = HistogramAccumulator.continuous("energy", 0.0, 4.0, 4)
left.fill_continuous_sample([(0.5, 2.0)])
right.fill_continuous_sample([(2.5, 3.0)])
left.merge_in_place(right)
left.update_results()

snapshot = left.snapshot()
assert snapshot.sample_count == 2
assert snapshot.bins[0].sum_weights == 2.0
assert snapshot.bins[0].sum_weights_squared == 4.0
assert snapshot.bins[0].sum_weights / snapshot.sample_count == 1.0
assert snapshot.bins[2].sum_weights == 3.0
assert snapshot.bins[2].sum_weights_squared == 9.0
assert right.snapshot().sample_count == 0
assert len(left.rebin(2).snapshot().bins) == 2
```

`snapshot()` includes both committed and pending samples. `update_results()` moves pending values into the completed counters; it is not needed merely to inspect the current snapshot. A successful `merge_in_place` consumes only the donor's pending samples, so merge worker accumulators before committing them.

11.3 Inspect and evaluate an existing state

The following workflow assumes that `./state` contains a generated integrand. Evaluation points are integrand-specific: their length must match the selected integrand and any discrete dimensions. Use the generated reference for the exact single-sample and batch contracts.

```
from collections.abc import Sequence

from gammaloop import GammaLoopAPI

api = GammaLoopAPI(state_folder="./state", read_only_state=True)
api.run("display processes")

print(api.get_run_history())
runtime = api.get_default_runtime_settings()
print(runtime.get("integrator.n_start"))


def evaluate_one(point: Sequence[float]) -> complex:
    result = api.evaluate_sample(
        point,
        process_id=0,
        minimal_output=True,
    )
    return result.integrand_result
```

Verification tier: compile
The documentation checks compile this example as Python syntax. They do not import or execute the native module: a runtime check additionally needs a built GammaLoop package, a provisioned backend and license, an existing generated state, and a point with the correct dimension.

The run method uses the same command language as the CLI and can change the in-memory state, settings, and run history. `read_only_state=True` protects files inside the active state directory; it does not make the Python object immutable or automatically persist the session. Inspect the live session through `get_run_history`, `get_global_settings`, and `get_active_command_blocks`.

For structured inspection, `get_integrand_info` describes the selected backend and graph structure, while `get_integrand_settings` and `get_default_runtime_settings` return detached, read-only `SettingsValue` snapshots. Use `get(path)`, attribute access, indexing, or `to_dict()` to read them; modifying derived Python values does not update the live session.

11.4 Sample evaluation contract

All three supported interfaces use the same two input layouts:

- An **integration-space** row contains the unit-hypercube coordinates expected by the selected integrand and discrete dimensions. Its required length is integrand-specific.
- A **momentum-space** row is `[k1x, k1y, k1z, k2x, k2y, k2z, ...]`: exactly one spatial `(px, py, pz)` triplet per independent loop momentum. It contains neither loop-energy components nor external momenta; GammaLoop obtains the latter from the integrand's kinematics. Select momentum space explicitly with `--momentum-space`, `momentum_space=true`, or `momentum_space=True`.

Every momentum-space row must therefore have a coordinate count divisible by three, and its number of complete triplets must match the selected integrand, graph, or graph group. `approach` axes use the same layout and dimension as their midpoint. The CLI, structured Rust API, and Python API all reach the same input builder and reject incomplete triplets before numerical evaluation.

Precision selection is independent of the coordinate layout. With `use_arb_prec=false`, the configured stability ladder may evaluate and escalate through `f64`, `f128`, and arbitrary precision. Setting `use_arb_prec=true` forces arbitrary-precision (Arb) internal evaluation, using the configured Arb stability level when available and a default Arb level otherwise. The `-f` CLI shorthand has the same behavior. CLI output, Python numeric fields, and ordinary Rust `EvaluateSamples` results remain `f64`. Only Rust `EvaluateSamplesPrecise` retains the numeric type used by the selected stability level.

`evaluate_sample` returns one sample result and the observable bundle for that one-sample batch. `evaluate_samples` accepts a two-dimensional NumPy array and returns per-sample results plus a batch-global observable bundle. Per-sample weights, discrete coordinates, graph names, and orientations must have the same row count as the input batch when provided. Graph and orientation selection applies only to momentum-space evaluation.

Both the ordinary Rust and Python endpoints expose numeric results through an `f64` contract, even when `use_arb_prec=True` forces arbitrary-precision internal evaluation. Evaluation may warm the integrand and update in-memory caches or observable snapshots, including in a read-only-state session. Rust-only callers that must retain the active precision use `evaluate_sample_precise` and `evaluate_samples_precise`.

Point evaluation returns the integrand before the parameterization Jacobian. The kinematics, normalization, and weights guide defines the exact $I_{\text{returned}} J_{\text{parameterization}} w_{\text{MC}}$ relation used during integration and the correlated event-weight boundary.

For a complete, source-backed run card and scripts that expose event groups, cut metadata, selectors, and merged histogram snapshots, follow the events and observables guide.

11.5 Symbolic conversion helpers

The supported module-level helpers are `atom_to_canonical_string` for parsing and canonicalizing a Symbolica expression, `evaluate_graph_overall_factor` for evaluating a graph's symbolic prefactor, and `to_dots` for rewriting tensor contractions into Idenso dot-product notation. Their generated entries document accepted strings, return values, failure modes, and task-oriented examples.

11.6 CLI and settings reference

11.6.1 Shell completion

`--completions` emits a static script from the same Clap command tree as `--help`. Load it for the current shell or save it in the shell's normal completion directory:

```
# Bash for the current session
source <(./gammaloop --completions bash)

# Zsh for the current session
source <(./gammaloop --completions zsh)
```

Fish, Elvish, PowerShell, and Nushell are also supported values. For Fish, pipe the output to `source`; for Nushell, save the generated module and then source it. Bash and Fish output also registers the repository wrapper spelling `./gammaloop`.

Static and state-aware completion
The generated shell script covers executable subcommands, flags, enumerated values, and path hints. Completion inside the interactive GammaLoop prompt additionally knows the active state's processes, integrands, settings, model objects, graphs, cuts, and orientations. A static shell script cannot contain that changing session data.

Use `./gammaloop --help` and the generated CLI and settings reference for command names, aliases, flags, positional arguments, defaults, possible values, and setting paths. The tutorials explain how commands and settings combine into a persistent workflow. The logging and diagnostics guide covers startup precedence, selective tag filters, sink routing, release-build limitations, and copyable investigation patterns.

12 Logging and diagnostics

This guide explains the default tracing configuration used by the CLI and API at startup and gives copyable filters for narrowing a noisy calculation to the subsystem and work unit that matter.

> **Before you start**
> **Prerequisites:** a built GammaLoop CLI and a state or run card that reaches the behavior you want to inspect. Filter changes take effect immediately and add negligible setup time, but verbose `inspect` or `dump` events can produce large files and slow a run. The expected invariant is that a narrow directive changes diagnostic visibility, not numerical results. If a filter produces nothing, first use a broader target at `debug`, confirm that release compilation has not removed the callsite, and then add tag constraints. After isolating the event, record the exact directive with the run card and continue to the CLI/settings reference for persistence and state controls.

12.1 Default directives

Normal session defaults come from `GlobalSettings`:

- `display_directive = "info"`
- `logfile_directive = "off"`

These are the application defaults written into `global_settings.toml` when defaults are shown or persisted.

In practice, `info` is the current convention for normal screen-facing output. Selective internal diagnostics should generally stay on `debug` and be turned on via narrower target/tag filters rather than by raising the whole display logger.

12.2 Startup precedence

Startup applies logging in this order:

1. Base settings come from `GlobalSettings`.
2. Environment overrides replace the base spec:
   - `GL_ALL_LOG_FILTER` overrides both stderr and logfile filters.
   - `GL_DISPLAY_FILTER` overrides only the stderr/display filter.
   - `GL_LOGFILE_FILTER` overrides only the logfile filter.
3. CLI overrides are then applied:
   - `-l/--level` overrides the stderr/display filter for the session.
   - `-L/--logfile-level` overrides the logfile filter for the session.
4. Some CLI modes hard-disable logfile logging for the session:
   - `--read-only-state`
   - `--logfile-level off`

When logfile logging is hard-disabled at boot, later settings changes cannot re-enable it for that session.

12.3 CLI level mapping

`-l/--level` and `-L/--logfile-level` expand to explicit crate directives:

- `off` -> `gammaloop_api=off,gammalooprs=off`
- `error` -> `gammaloop_api=error,gammalooprs=error`
- `warn` -> `gammaloop_api=warn,gammalooprs=warn`
- `info` -> `gammaloop_api=info,gammalooprs=info`

- `debug` -> `gammaloop_api=debug,gammalooprs=debug`
- `trace` -> `gammaloop_api=trace,gammalooprs=trace`

12.4 Empty-spec fallback floors

The filter builders also have hardcoded fallback floors used when a spec is empty:

- display/stderr builder fallback: `off`
- logfile builder fallback: `warn`

This is separate from the normal application defaults above. In other words:

- if startup uses the normal settings path, the defaults are `info` for display and `off` for logfile
- if an empty filter string is explicitly parsed, the display builder falls back to `off` and the logfile builder falls back to `warn`

12.5 Runtime updates

Changing `global.display_directive` or `global.logfile_directive` after startup reloads the active tracing filters. The CLI stderr override from `-l/--level` is treated as a session override and is separate from the persisted global setting.

12.6 Tag-based debug filtering

For selective debug logging, GammaLoop uses its own logging DSL rather than raw `EnvFilter`.

This follows the general idea in Tom Mrazik's tag-based logging note.

The main reason for owning the DSL is that GammaLoop needs composable tag groups and stable semantics around presence, absence, and explicit boolean values.

- Use stable targets for broad subsystems such as `gammalooprs::uv::forest` or `gammalooprs::integrands::process::cross_section`.
- Add boolean event fields as composable tags.
- Filter by field presence with `target[{tag_a,tag_b,!tag_c}]=debug`, or by displayed boolean values with `target[{#tag_a,#!tag_c}]=debug`.

This is preferable to span-name filtering because span-based filters can admit child-library events emitted while the span is active. GammaLoop tag filters only look at the event target and the boolean tag fields on the event itself.

12.6.1 Supported directive syntax

The supported directive forms are:

- `level`
- `target=level`
- `target[{tag_a,tag_b,!tag_c}]=level`
- `target[{#tag_a,#!tag_c}]=level`
- `target`

Notes:

- Directives are comma-separated.
- `target` is matched as a module-style prefix, so `gammalooprs::uv=debug` matches `gammalooprs::uv` and `gammalooprs::uv::forest`.
- A directive without an explicit `=level` is treated as `=trace`.
- Tag groups are conjunctions: `[{generation,uv}]` means both tags must match.
- `!tag` means the field is absent.
- `#tag` means the field is present with boolean value `true`.
- `#!tag` means the field is present with boolean value `false`.

- `field=value` matches an explicit field value.

12.6.2 Tag matching model

Tags are represented by event fields.

- `#generation` at the callsite emits `generation = true`.
- In a directive, `generation` means the event must carry a field named `generation`.
- In a directive, `#generation` means the event must carry `generation = true`.
- In a directive, `!inspect` means the event must not carry a field named `inspect`.
- In a directive, `#!inspect` means the event must carry `inspect = false`.
- In a directive, `inspect=true`, `mode=summary`, or `label="soft region"` means the event must carry that value specifically.

This means the callsite controls both:

- whether a log is selectable by a presence/absence query
- whether a log can be selected by an explicit value query such as `inspect=false` or `mode=summary`

12.6.3 Static vs dynamic filtering

Presence/absence-only directives stay on the lazy metadata path.

- `generation`
- `!inspect`
- `gammalooprs::uv::forest[{generation,uv,!dump}]=debug`

These can be decided from the callsite target and declared field names alone, so the filter can answer `always` or `never` at callsite registration time.

Value-matching directives are dynamic.

- `inspect=false`
- `#!inspect`
- `#generation`
- `mode=summary`
- `label="soft region"`

Those depend on the event instance, so they are evaluated in the per-event pass.

12.6.4 Precedence

When multiple directives match the same event, GammaLoop prefers the most specific one:

- longer target prefix wins
- then the directive with more tag requirements wins
- then later directives win if the earlier specificity is identical

This lets broad directives such as `gammalooprs=info` coexist with narrow tag-specific directives such as `gammalooprs::uv::forest[{generation,uv,dump}]=debug`.

12.6.5 Copy-pasteable display tags

Display logs render boolean fields next to the source as a directive tag group:

```
@gammalooprs::uv::forest[{#generation, #uv, #!inspect}] DEBUG: message
```

Every boolean field is rendered this way, not only the standard tag vocabulary. `true` becomes `#field_name`; `false` becomes `#!field_name`. These boolean fields are removed from the normal display field table.

Use the `full` logging prefix when you want the rendered source to be a valid directive head:

```
gammaloop --logging-prefix full -l debug ...
```

Then the text after `@` and before the log level can be copied into a directive by adding `=debug`, for example:

```
display_directive = "gammalooprs::uv::forest[{#generation, #uv, #!inspect}]=debug"
```

This copy-paste form assumes the display source is the module target. Enabling `full_line_source` adds file and line information to the source display, which is useful for locating code but is not a valid directive target.

12.6.6 Sink-only field prefixes

This repo also has sink-routing prefixes for fields:

- `file.<name>`: only rendered into the logfile/json sink
- `display.<name>`: only rendered into the stderr/display sink

Examples:

- `file.integrands = %...` will appear only in the file logger output
- `display.progress = %...` will appear only in the display logger output

These prefixes are a formatting/routing feature, not a separate event kind. They decide where a field is rendered after the event has been emitted. This lets a callsite attach large payloads such as `file.integrands` without dumping them to stderr.

12.6.7 Pipeline-wide tag set

Prefer a small stable vocabulary that cuts across the whole pipeline.

Primary phase tags:

- `generation`: work that builds or prepares runtime objects before Monte Carlo integration starts
- `integration`: work performed while evaluating samples or managing the adaptive integration loop
- `profile`: UV/IR/profile-style diagnostic scans and their analysis
- `persistence`: reading or writing saved state, manifests, checkpoints, and exported results

Core domain tags:

- `uv`: ultraviolet counterterms, UV forests, UV profiles, or UV-specific generation/evaluation logic
- `ir`: infrared subtraction, IR profiles, or IR-specific generation/evaluation logic
- `subtraction`: threshold, UV, or IR subtraction logic when the main concern is subtraction rather than the specific regime
- `sampling`: channel choice, discrete axes, parameterizations, or sample-generation choices
- `stability`: precision escalation, retries, instability diagnosis, and numerical safety checks
- `observables`: histogramming, event-to-observable projection, and observable snapshot production
- `selectors`: event-selection logic and selector decisions
- `cache`: cache lookup, reuse, invalidation, or cache-debug instrumentation

Common work-unit tags:

- `graph`: the log is about one graph or graph-local data
- `group`: the log is about a graph group or another explicitly grouped aggregate
- `orientation`: the log is about orientation-dependent data or choosing/summing orientations
- `channel`: the log is about multi-channeling or a particular channel
- `cut`: the log is about a cut, raised cut, or cut-local computation
- `event`: the log is about generated/retained event objects or event-group processing
- `sample`: the log is about one evaluation sample, its coordinates, or per-sample intermediate values
- `iteration`: the log is about one adaptive integration iteration or iteration-level summaries
- `term`: the log is about one algebraic term, summand, or term-local contribution

Common purpose tags:

- `solver`: the log is about root-finding, linear solves, fitting, or similar numerical solver state
- `compile`: the log is about evaluator/code generation or compilation-oriented preparation
- `inspect`: the log exposes detailed intermediate state for debugging, rather than a high-level milestone
- `summary`: the log is a compact roll-up rather than a step-by-step trace
- `dump`: the log emits large or structured payloads such as expressions, tables, or serialized views

12.6.8 Tag boundaries

Use the most general tag that accurately captures the reason you want to turn the log on or off.

- Prefer `generation` over `compile` when the message is a broad generation milestone.
- Add `compile` only when the message is specifically about evaluator construction or compilation-like work.
- Prefer `uv` or `ir` when the regime matters to the user; use `subtraction` when the subtraction mechanism is the real concern.
- Prefer `inspect` for verbose intermediate values that are mainly useful while debugging internals.
- Add `dump` when the payload is large enough that users may want to suppress it separately from lighter debug logs.
- Do not use `graph`, `cut`, or `sample` as substitutes for `generation` or `integration`; they refine phase tags rather than replace them.

12.6.9 Naming guidance

- Use phase tags first. They answer "when in the pipeline did this happen?"
- Use domain tags second. They answer "what subsystem is this about?"
- Use work-unit tags third. They answer "what object is being worked on?"
- Use purpose tags last for optional refinement.
- Prefer tags that are meaningful across multiple modules.
- Avoid tags that merely restate a function name.
- Avoid tags tied to one internal representation unless that representation is a stable concept in user-facing debugging.

`parametric` is usually not a core pipeline tag. It is acceptable as a local refinement when needed, but should not be treated as part of the primary vocabulary unless it becomes a consistently useful cross-cutting concept.

12.6.10 Example queries

- all generation-time UV forest logs:
  - `gammalooprs::uv::forest[{generation,uv}]=debug`
- only the large UV forest dumps for orientation-dependent generation logs:
  - `gammalooprs::uv::forest[{generation,uv,orientation,dump}]=debug`
- UV forest term-by-term generation logs:
  - `gammalooprs::uv::forest[{generation,uv,term}]=debug`
- all integration-time subtraction debug in amplitude evaluation:
  - `gammalooprs::integrands::process::amplitude[{integration,subtraction}]=debug`
- per-sample integration inspection in cross-section evaluation:
  - `gammalooprs::integrands::process::cross_section[{integration,sample,inspect}]=debug`
- all integration-time cut solver debug in cross-section evaluation:
  - `gammalooprs::integrands::process::cross_section[{integration,cut,solver}]=debug`
- integration-time debug excluding inspect-tagged logs:
  - `gammalooprs::integrands::process::cross_section[{integration,!inspect}]=debug`
- only events that explicitly set `inspect=true`:
  - `gammalooprs::integrands::process::cross_section[{integration,inspect=true}]=debug`
- only events with a specific textual mode field:
  - `gammalooprs::integrands::process::cross_section[{integration,mode=summary}]=debug`

Example display directive:

```
[cli_settings.global]
display_directive =
"gammaloop_api=info,gammalooprs=info,symbolica=off,poly::gcd=off,gammalooprs::uv::forest[{generation,uv,o
```

12.7 Unsupported `EnvFilter` syntax

GammaLoop no longer treats the directive string as generic `EnvFilter` syntax.

In particular, do not rely on:

- span-name filters such as `target[span_name]=debug`
- generic field filters such as `[{field=value}]`
- `EnvFilter` regex semantics

If those are needed later, they must be added explicitly to the GammaLoop DSL.

12.8 Release-build note

`gammalooprs` is compiled with `tracing` feature `release_max_level_info`, so `debug!` and `trace!` callsites in that crate are compiled out in release builds.

12.9 Authoritative implementation

The startup precedence and sink wiring live in the API tracing setup. The shared directive parser, CLI level mapping, and filter behavior live in the runtime tracing utilities, while persisted defaults are owned by GlobalSettings.

13 Kurvst curve and path API

Kurvst is GammaLoop's Typst-facing WebAssembly layer for Bezier and Hobby paths, arc-length trimming, sampled path patterns, parallel curves, and CeTZ conversion. This page embeds the complete maintained Kurvst manual used by the drawing pipeline.

> **Geometry rather than topology**
> Kurvst transforms curve geometry. Linnet and Linnest own graph topology and layout; GammaLoop's drawing templates combine those results with Kurvst paths and styles.

`kurvst` is a Typst package backed by `kurvst.wasm`, a small Kurbo-based plugin. Its public API is path-based: construct a path dictionary, pass that path into transforms, and draw the returned path.

It exposes Bezier utilities that are awkward or unavailable in native Typst:

- constructing cubic and Hobby paths,
- trimming paths by arc length,
- generating sampled path patterns,
- fitting Kurbo offset/parallel paths,
- converting returned path geometry to CeTZ drawing commands.

13.1 Choose an import path

Kurvst is currently a bundled source package, not a Typst Universe package. A Clinnet run writes it below `build/templates/crates/kurvst/typst/`; a custom template in `build/templates/` can therefore import it with:

```
#import "crates/kurvst/typst/src/lib.typ" as kurvst
```

Repository examples live one directory beside `src/` and instead use `#import "../src/lib.typ" as kurvst`. Keep `kurvst.wasm`, `typst.toml`, and the `src/` tree together when copying the package to another Typst project.

Path geometry points are two-item tuples:

```
#let p = (1.2, -0.4)
```

Cubic segments use `start`, `control-start`, `control-end`, and `end`:

```
#let segment = (
  start: (0, 0),
  control-start: (1, 0.5),
  control-end: (2, -0.5),
  end: (3, 0),
)
```

Use `from-cubic(segment)` once to turn that cubic dictionary into a path dictionary. Returned paths contain a curve-command wire path in `path`. Rust converts that wire path to Kurbo's `BezPath` internally, and every path-level Kurvst function accepts either a returned dictionary or the raw wire path.

```
#let path = kurvst.from-cubic(segment)
#let trimmed = kurvst.trim(path, start-outset: 0.2, end-outset: 0.1)
#let shifted = kurvst.parallel(trimmed, distance: 0.15)
```

13.2 Path Wire Format

A Kurvst wire path is a dictionary with one `elements` array. The elements mirror Typst's native `curve` commands, but points are plain numeric tuples:

```
#let path = (
  elements: (
    (kind: "move", start: (0, 0)),
    (kind: "line", end: (0.6, 0.3)),
    (kind: "quad", control: (1.0, 1.0), end: (1.4, 0.3)),
    (
      kind: "cubic",
      control-start: (1.8, -0.4),
      control-end: (2.4, 1.0),
      end: (3.0, 0),
    ),
    (kind: "close", mode: "straight"),
  ),
)
```

The supported element kinds are:

- `move`: starts a subpath at `start`.
- `line`: adds a straight segment ending at `end`.
- `quad`: adds a quadratic segment using `control` and `end`.
- `cubic`: adds a cubic segment using `control-start`, `control-end`, and `end`.
- `close`: closes the current subpath. `mode` is optional and defaults to `"straight"` when emitted by Kurvst.

Prefer the path fragment helpers for new code. `line`, `quad`, and `cubic` include their start points, so fragments can be built independently:

```
#let custom = kurvst.path(
  kurvst.line((0, 0), (0.6, 0.3)),
```

```
  kurvst.quad((0.6, 0.3), (1.0, 1.0), (1.4, 0.3)),
  kurvst.cubic((1.4, 0.3), (1.8, -0.4), (2.4, 1.0), (3.0, 0)),
)
```

`path` and `append` flatten fragments. If an appended fragment starts where the current path ends, its leading `move` is skipped; if it starts elsewhere, the `move` begins a new subpath. The edited path can go straight back into Kurvst transforms:

```
#let base = kurvst.from-cubic(segment)
#let extended = kurvst.append(base, kurvst.line(segment.end, (3.4, 0.2)))

#let trimmed = kurvst.trim(extended, start-outset: 0.15)
#let shifted = kurvst.parallel(trimmed, distance: 0.1)

#kurvst.to-native(shifted, unit: 36pt, stroke: rgb("#1b7f4c") + 0.7pt)
```

Use `elements` when you need the raw command stream, `points` for the visited endpoints, `segments` for cubic segment dictionaries, `to-native` for native Typst drawing, and `to-cetz-data` or `to-cetz` for CeTZ.

A typical chain keeps the returned dictionaries intact until the final drawing step:

```
#let base = kurvst.hobby-through(demo-start, demo-through, demo-end)
#let trimmed = kurvst.trim(base, start-outset: 0.2, end-outset: 0.1)
#let shifted = kurvst.parallel(trimmed, distance: 0.15)

#cetz.canvas({
  kurvst.to-cetz(base, stroke: rgb("#bbbbbb") + 0.4pt)
  kurvst.to-cetz(trimmed, stroke: rgb("#d72638") + 0.6pt)
  kurvst.to-cetz(shifted, stroke: rgb("#1b7f4c") + 0.6pt)
})
```

13.3 Paths And Trimming

`cubic` constructs a path from one cubic Bezier. `trim` trims by curve distance from the start and/or end and returns another path dictionary.

```
#let path = kurvst.from-cubic(segment)
#let trimmed = kurvst.trim(path, start-outset: 0.15, end-outset: 0.1)
```

`hobby-through(start, through, end)` builds a smooth open curve through three points; `segments` returns the two cubic halves split at `through`. `hobby-spline(points)` does the same for any open point sequence with at least two points. Both return path dictionaries and can be passed directly to path-level helpers.

```
#let spline = kurvst.hobby-spline((
  (0.0, 0.0),
  (0.9, 0.8),
  (1.8, -0.3),
  (2.8, 0.4),
))
#let shifted = kurvst.parallel(spline, distance: 0.14)
#let wiggle = kurvst.pattern(
  spline,
  pattern: kurvst.wave(samples-per-period: 24),
  amplitude: 0.1,
  wavelength: 0.55,
)
```

```
#cetz.canvas({
  kurvst.to-cetz(spline, stroke: rgb("#bbbbbb") + 0.45pt)
  kurvst.to-cetz(shifted, stroke: rgb("#1b7f4c") + 0.6pt)
  kurvst.to-cetz(wiggle, stroke: rgb("#355c9a") + 0.55pt)
})
```

`split-through` combines Hobby spline construction with endpoint trimming and returns one path part between each consecutive point. Consumers such as Linnest can wrap it for graph-edge geometry.

13.4 Path Patterns

`pattern` generates a one-dimensional pattern along any path dictionary. Built-ins are regular Typst dictionaries:

```
#let wave = kurvst.wave()
#let zigzag = kurvst.zigzag()
#let coil = kurvst.coil(longitudinal-scale: 1.6)
```

String names `"wave"`, `"zigzag"`, and `"coil"` are accepted for convenience, but they are resolved in Typst before calling wasm.

Custom patterns use the same object shape:

```
#let hook = (
  kind: "points",
  name: "hook",
  interpolation: "linear",
  points: (
    (at: 0, x: 0, y: 0),
    (at: 0.25, x: 0.15, y: 1),
    (at: 0.5, x: 0, y: 0),
    (at: 0.75, x: -0.15, y: -1),
    (at: 1, x: 0, y: 0),
  ),
)
```

`at` is the phase position inside one wavelength, from `0` to `1`. The `x` coordinate offsets along the path tangent and `y` offsets along the path normal; both are scaled by `amplitude`. If `at` is omitted, points are spaced evenly. Use `interpolation: "linear"` for corners and `"smooth"` for a spline through sampled points.

`endpoint-ramp: true` tapers amplitude at anchored endpoints. This is useful for coils, whose longitudinal offset would otherwise put the first and last visible points inside the turn near nodes.

```
#let base = kurvst.from-cubic(segment)
#let path = kurvst.pattern(
  base,
  pattern: hook,
  amplitude: 0.15,
  wavelength: 0.7,
)
#native-scene({
  native-cubics(kurvst.segments(base), stroke: rgb("#c8c8c8") + 0.45pt)
  native-polyline(kurvst.points(path), stroke: rgb("#1b7f4c") + 0.8pt)
})
```

13.5 Parallel Paths

`parallel` uses Kurbo's offset curve fitter to produce a path at a fixed normal distance from the source path. Positive distances follow the left normal of the path direction; negative distances follow the right normal. `start-outset` and `end-outset` trim the fitted path by arc length after the offset is computed.

```
#let base = kurvst.from-cubic(segment)
#let left = kurvst.parallel(base, distance: 0.18, start-outset: 0.35, end-outset: 0.35)
#let right = kurvst.parallel(base, distance: -0.18)
#native-scene({
  native-cubics(kurvst.segments(base), stroke: rgb("#c8c8c8") + 0.45pt)
  native-cubics(kurvst.segments(left), stroke: rgb("#1b7f4c") + 0.8pt)
  native-cubics(kurvst.segments(right), stroke: rgb("#355c9a") + 0.8pt)
})
```

13.6 Path Layers

`layer` is a convenience wrapper for drawing derived visible paths. It combines side-aware offsetting, endpoint outsets, and centered shortening into one path-in/path-out operation. `length` is a fixed visible arc length, `ratio` is a fraction of the input path length, and `resolve-length` decides how to combine them. The default `"min"` keeps whichever limit is shorter.

```
#let base = kurvst.hobby-spline((
  (0.0, 0.0),
  (0.9, 0.8),
  (1.8, -0.3),
  (2.8, 0.4),
))
#let layer = kurvst.layer(
  base,
  offset: 0.16,
  length: 1.6,
  ratio: 0.5,
  resolve-length: "min",
)
#cetz.canvas({
  kurvst.to-cetz(base, stroke: rgb("#bbbbbb") + 0.45pt)
  kurvst.to-cetz(layer, stroke: rgb("#1b7f4c") + 0.8pt)
})
```

Use `side-point` to choose the sign of the offset from a point on the desired side of the path. Drawing packages can use this to place derived layers on the same side as an edge label without knowing anything about the physics or graph style that requested the layer.

13.7 Native Drawing Primitives

Kurvst returns dictionaries and arrays. The core geometry can be drawn without CeTZ by emitting native `curve` content:

```
#kurvst.to-native(kurvst.from-cubic(segment),
unit: 36pt, stroke: black + 0.7pt)
```

The generated reference below keeps drawing code inline. It only uses the small common helpers above for coordinate scaling, native `curve` construction, point markers, and fixed-size preview blocks.

13.8 CeTZ Interoperability

The CeTZ helpers are thin adapters over the same returned geometry. Use them when the surrounding document already lives in a CeTZ canvas, or when you want CeTZ path merging and styling:

```
#cetz.canvas({
  let base = kurvst.from-cubic(segment)
  kurvst.to-cetz(base, stroke: rgb("#d72638")
+ 0.6pt)

  let path = kurvst.pattern(
    base,
    pattern: kurvst.coil(longitudinal-scale:
1.6),
    amplitude: 0.12,
    wavelength: 0.55,
  )
  kurvst.to-cetz(path, stroke: rgb("#355c9a")
+ 0.55pt)
})
```

13.9 Generated Reference

13.10 kurvst

- append()
- center-outset()
- close()
- coil()
- cubic()
- cubic-point()
- cubic-tangent()
- cubic-to()
- elements()
- from-cubic()
- from-elements()
- hobby-spline()
- hobby-through()
- layer()
- length()
- line()
- line-segment()
- line-to()
- move-to()
- outset-point()
- parallel()
- path()
- pattern()
- point()
- points()
- quad()
- quad-to()
- resolve-length()
- segments()
- split-through()
- to-cetz()

- to-cetz-data()
- to-native()
- trim()
- wave()
- zigzag()

Variables
- layer-defaults

13.10.1 append

Return a path with additional fragments or elements appended.

13.10.1.1 Parameters

```
append(
  path: dictionary,
  ..parts: any
) -> dictionary
```

13.10.1.1.1 path `dictionary`

Base Kurvst path dictionary.

13.10.1.1.2 ..parts `any`

Path fragments, path elements, or element arrays to append.

13.10.2 center-outset

Compute the symmetric trim needed to center a shorter path layer.

13.10.2.1 Parameters

```
center-outset(
  base-length: int float,
  length: none int float,
  ratio: none int float,
  resolve-length: string function,
  start-outset: int float,
  end-outset: int float
) -> int float
```

13.10.2.1.1 base-length `int` or `float`

Full base path arc length.

13.10.2.1.2 length `none` or `int` or `float`

Fixed target visible length.

Default: `none`

13.10.2.1.3 ratio `none` or `int` or `float`

Relative target visible length as a fraction of `base-length`.

Default: `none`

13.10.2.1.4 resolve-length `string` or `function`

Resolution strategy for fixed and relative targets.

Default: `"min"`

13.10.2.1.5 start-outset `int` or `float`

Already-applied trim at the start of the path.

Default: `0`

13.10.2.1.6 end-outset `int` or `float`

Already-applied trim at the end of the path.

Default: `0`

13.10.3 close

Build a `close` path element.

13.10.3.1 Parameters

`close(mode: string) -> dictionary`

13.10.3.1.1 mode `string`

Native Typst curve close mode.

Default: `"straight"`

13.10.4 coil

A smooth coil path pattern.

13.10.4.1 Parameters

```
coil(
  samples-per-period: int,
  longitudinal-scale: int float
) -> dictionary
```

13.10.4.1.1 samples-per-period `int`

Number of samples used to approximate one coil period.

Default: `16`

13.10.4.1.2 longitudinal-scale `int` or `float`

Horizontal scale of the coil before it is mapped onto a path.

Default: `1.25`

13.10.5 cubic

Build a cubic path fragment.

13.10.5.1 Parameters

```
cubic(
  start: array,
  control-start: array,
  control-end: array,
  end: array
) -> dictionary
```

13.10.5.1.1 start `array`

Start point.

13.10.5.1.2 control-start `array`

Cubic control point near `start`.

13.10.5.1.3 control-end `array`

Cubic control point near `end`.

13.10.5.1.4 end `array`

Endpoint.

13.10.6 cubic-point

Evaluate a cubic segment at parameter `t`.

13.10.6.1 Parameters

```
cubic-point(
  segment: dictionary,
  t: int float
) -> array
```

13.10.6.1.1 segment `dictionary`

Segment with `start`, `control-start`, `control-end`, and `end`.

13.10.6.1.2 t `int` or `float`

Segment parameter in the range `[0, 1]`.

13.10.7 cubic-tangent

Evaluate the tangent of a cubic segment at parameter `t`.

13.10.7.1 Parameters

```
cubic-tangent(
  segment: dictionary,
  t: int float
) -> array
```

13.10.7.1.1 segment `dictionary`

Segment with `start`, `control-start`, `control-end`, and `end`.

13.10.7.1.2 t `int` or `float`

Segment parameter in the range `[0, 1]`.

13.10.8 cubic-to

Build a `cubic` path element.

13.10.8.1 Parameters

```
cubic-to(
  control-start: array,
  control-end: array,
  end: array
) -> dictionary
```

13.10.8.1.1 control-start `array`

Cubic control point near the start point.

13.10.8.1.2 control-end `array`

Cubic control point near the endpoint.

13.10.8.1.3 end `array`

Cubic endpoint.

13.10.9 elements

Return the command elements that make up a Kurvst path.

13.10.9.1 Parameters

```
elements(path: dictionary) -> array
```

13.10.9.1.1 path `dictionary`

Kurvst path dictionary to inspect.

13.10.10 from-cubic

Build a path fragment from a cubic segment dictionary.

13.10.10.1 Parameters

```
from-cubic(segment: dictionary) -> dictionary
```

13.10.10.1.1 segment `dictionary`

Segment with `start`, `control-start`, `control-end`, and `end`.

13.10.11 from-elements

Build a path dictionary from an existing element array.

13.10.11.1 Parameters

```
from-elements(elements: array) -> dictionary
```

13.10.11.1.1 elements `array`

Array of Kurvst path elements.

13.10.12 hobby-spline

Construct a Hobby spline through an arbitrary point sequence.

13.10.12.1 Parameters

```
hobby-spline(
  points: array,
  omega: float,
  accuracy: float
) -> dictionary
```

13.10.12.1.1 points `array`

Two or more points for the open spline to pass through.

13.10.12.1.2 omega `float`

Hobby curl/tension parameter.

Default: `1.0`

13.10.12.1.3 accuracy `float`

Geometry approximation accuracy passed to the Rust geometry engine.

Default: `0.001`

13.10.13 hobby-through

Construct a cubic Hobby path through three points.

13.10.13.1 Parameters

```
hobby-through(
  start: array,
  through: array,
  end: array,
  omega: float,
  accuracy: float
) -> dictionary
```

13.10.13.1.1 start `array`

Start point.

13.10.13.1.2 through `array`

Intermediate point that the curve passes through.

13.10.13.1.3 end `array`

Endpoint.

13.10.13.1.4 omega `float`

Hobby curl/tension parameter.

Default: `1.0`

13.10.13.1.5 accuracy `float`

Geometry approximation accuracy passed to the Rust geometry engine.

Default: `0.001`

13.10.14 layer

Build a derived visible path layer.

13.10.14.1 Parameters

```
layer(
  path: dictionary,
  offset: int float,
  length: none int float,
  ratio: none int float,
  resolve-length: string function,
  start-outset: int float,
  end-outset: int float,
  side-point: none array,
  accuracy: float,
  optimize: bool
) -> dictionary
```

13.10.14.1.1 path `dictionary`

Base Kurvst path dictionary.

13.10.14.1.2 offset `int` or `float`

Signed normal offset distance.

Default: `0`

13.10.14.1.3 length `none` or `int` or `float`

Fixed target visible length.

Default: `none`

13.10.14.1.4 ratio `none` or `int` or `float`

Relative target visible length as a fraction of the base length.

Default: `none`

13.10.14.1.5 resolve-length `string` or `function`

Resolution strategy for fixed and relative targets.

Default: `"min"`

13.10.14.1.6 start-outset `int` or `float`

Arc length removed from the start.

Default: `0`

13.10.14.1.7 end-outset `int` or `float`

Arc length removed from the end.

Default: `0`

13.10.14.1.8 side-point `none` or `array`

Optional point used to choose the sign of `offset`.

Default: `none`

13.10.14.1.9 accuracy `float`

Geometry approximation accuracy passed to the Rust geometry engine.

Default: `0.001`

13.10.14.1.10 optimize `bool`

Let Kurbo simplify/optimize fitted parallel paths.

Default: `true`

13.10.15 length

Compute the arc length of a path dictionary.

13.10.15.1 Parameters

```
length(
  path: dictionary,
  accuracy: float
) -> int float
```

13.10.15.1.1 path `dictionary`

Kurvst path dictionary to measure.

13.10.15.1.2 accuracy `float`

Arc-length approximation accuracy passed to the Rust geometry engine.

Default: `0.001`

13.10.16 line

Build a straight-line path fragment.

13.10.16.1 Parameters

```
line(
  start: array,
  end: array
) -> dictionary
```

13.10.16.1.1 start `array`

Start point.

13.10.16.1.2 end `array`

Endpoint.

13.10.17 line-segment

Build a cubic segment dictionary for a straight line.

13.10.17.1 Parameters

```
line-segment(
  start: array,
  end: array
) -> dictionary
```

13.10.17.1.1 start `array`

Start point.

13.10.17.1.2 end `array`

Endpoint.

13.10.18 line-to

Build a `line` path element.

13.10.18.1 Parameters

```
line-to(end: array) -> dictionary
```

13.10.18.1.1 end `array`

Line endpoint.

13.10.19 move-to

Build a `move` path element.

13.10.19.1 Parameters

```
move-to(start: array) -> dictionary
```

13.10.19.1.1 start `array`

New current point and subpath start.

13.10.20 outset-point

Return `from` moved toward `toward` by `distance`.

13.10.20.1 Parameters

```
outset-point(
  from: array,
  toward: array,
  distance: int float
) -> array
```

13.10.20.1.1 from `array`

Point to move.

13.10.20.1.2 toward `array`

Target point that defines the direction.

13.10.20.1.3 distance `int` or `float`

Distance to move from `from` toward `toward`.

Default: `0`

13.10.21 parallel

Generate a parallel path for a path.

13.10.21.1 Parameters

```
parallel(
  path: dictionary,
  distance: int float,
  start-outset: int float,
  end-outset: int float,
  accuracy: float,
  optimize: bool
) -> dictionary
```

13.10.21.1.1 path `dictionary`

Base Kurvst path dictionary.

13.10.21.1.2 distance `int` or `float`

Signed normal offset distance.

Default: `0`

13.10.21.1.3 start-outset `int` or `float`

Arc length removed from the start before offsetting.

Default: `0`

13.10.21.1.4 end-outset `int` or `float`

Arc length removed from the end before offsetting.

Default: `0`

13.10.21.1.5 accuracy `float`

Geometry approximation accuracy passed to the Rust geometry engine.

Default: `0.001`

13.10.21.1.6 optimize `bool`

Let Kurbo simplify/optimize the fitted path.

Default: `true`

13.10.22 path

Build a path dictionary from path fragments or elements.

13.10.22.1 Parameters

```
path(..parts: any) -> dictionary
```

13.10.22.1.1 ..parts `any`

Path fragments, path elements, or element arrays to concatenate.

13.10.23 pattern

Apply a repeated path pattern to a base path.

13.10.23.1 Parameters

```
pattern(
  path: dictionary,
  pattern: string dictionary,
  amplitude: int float,
  wavelength: int float,
  phase: int float,
  samples-per-period: int,
  coil-longitudinal-scale: int float,
  anchor-start: bool,
  anchor-end: bool,
  accuracy: float
) -> dictionary
```

13.10.23.1.1 path `dictionary`

Base Kurvst path dictionary.

13.10.23.1.2 pattern `string` or `dictionary`

Pattern dictionary or built-in pattern name.

Default: `"wave"`

13.10.23.1.3 amplitude `int` or `float`

Normal amplitude of the pattern in path units.

Default: `0.1`

13.10.23.1.4 wavelength `int` or `float`

Arc length of one pattern period.

Default: `1.0`

13.10.23.1.5 phase `int` or `float`

Initial phase offset in pattern periods.

Default: `0`

13.10.23.1.6 samples-per-period `int`

Samples per period for string-resolved smooth patterns.

Default: `16`

13.10.23.1.7 coil-longitudinal-scale `int` or `float`

Longitudinal scale used when resolving the built-in coil pattern.

Default: `1.25`

13.10.23.1.8 anchor-start `bool`

Force the generated path to start on the base path.

Default: `true`

13.10.23.1.9 anchor-end `bool`

Force the generated path to end on the base path.

Default: `true`

13.10.23.1.10 accuracy `float`

Geometry approximation accuracy passed to the Rust geometry engine.

Default: `0.001`

13.10.24 point

Build a numeric point tuple.

13.10.24.1 Parameters

```
point(
  x: int float,
  y: int float
) -> array
```

13.10.24.1.1 x `int` or `float`

X coordinate.

13.10.24.1.2 y `int` or `float`

Y coordinate.

13.10.25 points

Return the points visited by a Kurvst path's command stream.

13.10.25.1 Parameters

```
points(path: dictionary) -> array
```

13.10.25.1.1 path `dictionary`

Kurvst path dictionary to inspect.

13.10.26 quad

Build a quadratic path fragment.

13.10.26.1 Parameters

```
quad(
  start: array,
  control: array,
  end: array
) -> dictionary
```

13.10.26.1.1 start `array`

Start point.

13.10.26.1.2 control `array`

Quadratic control point.

13.10.26.1.3 end `array`

Endpoint.

13.10.27 quad-to

Build a quad path element.

13.10.27.1 Parameters

```
quad-to(
  control: array,
  end: array
) -> dictionary
```

13.10.27.1.1 control `array`

Quadratic control point.

13.10.27.1.2 end `array`

Quadratic endpoint.

13.10.28 resolve-length

Resolve a fixed and relative visible path length.

13.10.28.1 Parameters

```
resolve-length(
  base-length: int float,
  length: none int float,
  ratio: none int float,
  method: string function
) -> none int float
```

13.10.28.1.1 base-length `int` or `float`

Full base path arc length.

13.10.28.1.2 length `none` or `int` or `float`

Fixed target arc length.

Default: none

13.10.28.1.3 ratio `none` or `int` or `float`

Relative target length as a fraction of `base-length`.

Default: `none`

13.10.28.1.4 method `string` or `function`

Resolution strategy for fixed and relative targets.

Default: `"min"`

13.10.29 segments

Return drawable cubic segments for any Kurvst path dictionary.

13.10.29.1 Parameters

```
segments(path: dictionary) -> array
```

13.10.29.1.1 path `dictionary`

Kurvst path dictionary to convert.

13.10.30 split-through

Split a path through a point sequence into per-span paths.

13.10.30.1 Parameters

```
split-through(
  points: array,
  omega: float,
  start-outset: int float,
  end-outset: int float,
  accuracy: float
) -> dictionary
```

13.10.30.1.1 points `array`

Two or more points for the curve to pass through.

13.10.30.1.2 omega `float`

Hobby curl/tension parameter.

Default: `1.0`

13.10.30.1.3 start-outset `int` or `float`

Arc length removed from the first span start.

Default: `0`

13.10.30.1.4 end-outset `int` or `float`

Arc length removed from the last span end.

Default: `0`

13.10.30.1.5 accuracy `float`

Geometry approximation accuracy passed to the Rust geometry engine.

Default: `0.001`

13.10.31 to-cetz

Draw a path dictionary through CeTZ.

13.10.31.1 Parameters

```
to-cetz(
  path: dictionary,
  unit: int float length ratio,
  ..style: any
) -> content
```

13.10.31.1.1 path `dictionary`

Kurvst path dictionary to draw.

13.10.31.1.2 unit `int` or `float` or `length` or `ratio`

Coordinate multiplier for emitted CeTZ points.

Default: `1`

13.10.31.1.3 ..style `any`

CeTZ draw style arguments forwarded to `merge-path`.

13.10.32 to-cetz-data

Emit a Kurvst path as CeTZ path data.

13.10.32.1 Parameters

```
to-cetz-data(
  path: dictionary,
  unit: int float length ratio
) -> array
```

13.10.32.1.1 path `dictionary`

Kurvst path dictionary to convert.

13.10.32.1.2 unit `int` or `float` or `length` or `ratio`

Coordinate multiplier for emitted CeTZ points.

Default: 1

13.10.33 to-native

Emit a Kurvst path as native Typst `curve` content.

13.10.33.1 Parameters

```
to-native(
  path: dictionary,
  unit: int float length ratio,
  ..style: any
) -> content
```

13.10.33.1.1 path `dictionary`

Kurvst path dictionary to emit.

13.10.33.1.2 unit `int` or `float` or `length` or `ratio`

Coordinate multiplier for emitted Typst curve points.

Default: 1

13.10.33.1.3 ..style `any`

Native `curve` style arguments.

13.10.34 trim

Trim a path by arc length from each end.

13.10.34.1 Parameters

```
trim(
  path: dictionary,
  start-outset: int float,
  end-outset: int float,
  accuracy: float
) -> dictionary
```

13.10.34.1.1 path `dictionary`

Kurvst path dictionary to trim.

13.10.34.1.2 start-outset `int` or `float`

Arc length removed from the start.

Default: `0`

13.10.34.1.3 end-outset `int` or `float`

Arc length removed from the end.

Default: `0`

13.10.34.1.4 accuracy `float`

Arc-length approximation accuracy passed to the Rust geometry engine.

Default: `0.001`

13.10.35 wave

A smooth sinusoidal path pattern.

13.10.35.1 Parameters

```
wave(samples-per-period: int) -> dictionary
```

13.10.35.1.1 samples-per-period `int`

Number of samples used to approximate one wave period.

Default: `16`

13.10.36 zigzag

A straight-segment triangular path pattern.

13.10.36.1 Parameters

`zigzag() -> dictionary`

`dictionary`

13.10.37 layer-defaults

Default geometry options for derived path layers.

14 Version history

> **History coverage**
> GammaLoop does not yet have a consolidated changelog. Repository tags and their linked pull requests are the available release record. This page makes that gap explicit instead of presenting development documentation as a complete release history.

14.1 Components and version sources

- Repository revision: GammaLoop tags and releases.
- Python distribution: pyproject.toml.
- Core Rust library: gammalooprs package metadata.
- Rust/Python application API: gammaloop-api package metadata.

These components can evolve independently. Record all of them when reporting a result or asking for support.

14.2 Choosing the matching documentation

The `latest` channel follows current development. Documentation under `snapshots/<tag>` stays with a tagged release. Use the snapshot matching your checkout whenever command options, state files, or API signatures differ from `latest`.

14.3 Upgrade checklist

Release notes currently live with repository tags and their linked pull requests rather than in one consolidated changelog. Review them before upgrading, especially when you depend on persisted states, CLI scripts, Python bindings, or numerical results. Keep a copy of valuable state directories and re-run validation samples after changing versions.

14.4 Reproducibility record

Include the repository tag or commit, the Python distribution version, the `gammalooprs` version, the run card, and any relevant settings overrides. This makes a calculation easier to reproduce and distinguishes a software change from a configuration change.

Python API reference

The supported Python packages available in this version are listed below. The HTML manual provides a separate page for every public class and function; this printable edition keeps a compact inventory. Rust APIs use canonical Rustdoc in the HTML manual.

`gammaloop-python`

Exports registered by gammaloop.

`AdditionalWeight`

Named auxiliary complex weight attached to a generated event.

Named auxiliary complex weight attached to a generated event.

```
class AdditionalWeight:
```

Requires features: `python_api, python_abi, python_stubgen`

`key`

Kind: `Getter`

```
def key(self) -> str:
```

Stable weight name, optionally followed by colon-separated identifiers.

`value`

Kind: `Getter`

```
def value(self) -> Any:
```

Complex auxiliary event weight associated with ``key``.

Exhaustive reference · Source

`BatchEvaluationResult`

Per-sample evaluations paired with one observable snapshot for the complete batch.

Per-sample evaluations paired with one observable snapshot for the complete batch.

```
class BatchEvaluationResult:
```

Requires features: `python_api, python_abi, python_stubgen`

`samples`

Kind: `Getter`

```
def samples(self) -> list[SampleEvaluationResult]:
```

Evaluation records in the same order as the requested samples.

`observables`

Kind: `Getter`

```
def observables(self) -> dict:
```

Observable snapshots accumulated across the complete sample batch.

`__str__`

Kind: `Method`

```
def __str__(self) -> str:
```

Return a human-readable batch and observable summary.

__repr__

Kind: Method

```
def __repr__(self) -> str:
```

Return the human-readable batch and observable summary.

Exhaustive reference · Source

CutInfo

Process, graph, orientation, and loop-basis identifiers for one cut event.

Process, graph, orientation, and loop-basis identifiers for one cut event.

```
class CutInfo:
```

Requires features: `python_api, python_abi, python_stubgen`

incoming_pdgs

Kind: Getter

```
def incoming_pdgs(self) -> list[int]:
```

PDG identifiers aligned with ``Event.incoming_momenta``.

outgoing_pdgs

Kind: Getter

```
def outgoing_pdgs(self) -> list[int]:
```

PDG identifiers aligned with ``Event.outgoing_momenta``.

cut_id

Kind: Getter

```
def cut_id(self) -> int:
```

Zero-based cut identifier within the selected graph.

graph_id

Kind: Getter

```
def graph_id(self) -> int:
```

Zero-based graph identifier within the integrand.

graph_group_id

Kind: Getter

```
def graph_group_id(self) -> Optional[int]:
```

Graph-group identifier, when group metadata is available.

orientation_id

Kind: Getter

```
def orientation_id(self) -> Optional[int]:
```

Causal-flow orientation identifier, when sampled explicitly.

lmb_channel_id

Kind: Getter

```
def lmb_channel_id(self) -> Optional[int]:
```

Loop-momentum-basis multichannel identifier, when sampled explicitly.

lmb_channel_edge_ids

Kind: Getter

```
def lmb_channel_edge_ids(self) -> Optional[list[int]]:
```

Edge identifiers defining the selected loop-momentum basis, when available.

Exhaustive reference · Source

DotExportSettings

Controls which graph details and algebraic transformations are included in DOT exports.

Controls which graph details and algebraic transformations are included in DOT exports.

```
class DotExportSettings:
```

Requires features: `python_api, python_abi, python_stubgen`

combine_diagrams

Kind: Getter

```
def combine_diagrams(self) -> bool:
```

Write all diagrams of an integrand to one file during filesystem export.

combine_diagrams

Kind: Setter

```
def combine_diagrams(self, value: bool) -> None:
```

Write all diagrams of an integrand to one file during filesystem export.

value

Kind: Parameter

```
bool
```

with_uv

Kind: Getter

```
def with_uv(self) -> bool:
```

Request UV counterterm graph output; currently retained for compatibility.

with_uv

Kind: Setter

```
def with_uv(self, value: bool) -> None:
```

Request UV counterterm graph output; currently retained for compatibility.

value

Kind: Parameter

```
bool
```

output_full_numerator

Kind: Getter

```
def output_full_numerator(self) -> bool:
```

Add the complete symbolic numerator as the graph-level `full_num` field.

output_full_numerator

Kind: Setter

```
def output_full_numerator(self, value: bool) -> None:
```

Add the complete symbolic numerator as the graph-level `full_num` field.

value

Kind: Parameter

```
bool
```

split_xs_by_initial_states

Kind: Getter

```
def split_xs_by_initial_states(self) -> bool:
```

Split cross-section graphs by distinct initial-state assignments.

split_xs_by_initial_states

Kind: Setter

```
def split_xs_by_initial_states(self, value: bool) -> None:
```

Split cross-section graphs by distinct initial-state assignments.

value

Kind: Parameter

```
bool
```

do_gamma_algebra

Kind: Getter

```
def do_gamma_algebra(self) -> bool:
```

Simplify gamma-matrix algebra in the full numerator before formatting it.

do_gamma_algebra

Kind: Setter

```
def do_gamma_algebra(self, value: bool) -> None:
```

Simplify gamma-matrix algebra in the full numerator before formatting it.

value

Kind: Parameter

```
bool
```

do_color_algebra

Kind: Getter

```
def do_color_algebra(self) -> bool:
```

Simplify color algebra in the full numerator before formatting it.

do_color_algebra

Kind: Setter

```
def do_color_algebra(self, value: bool) -> None:
```

Simplify color algebra in the full numerator before formatting it.

value

Kind: Parameter

bool

include_autogenerated_fields

Kind: Getter

def include_autogenerated_fields(self) -> bool:

Include vertex and edge fields whose values GammaLoop generated automatically.

include_autogenerated_fields

Kind: Setter

def include_autogenerated_fields(self, value: bool) -> None:

Include vertex and edge fields whose values GammaLoop generated automatically.

value

Kind: Parameter

bool

__new__

Kind: AssociatedFunction

def __new__(cls) -> DotExportSettings:

Construct DOT export settings with GammaLoop's defaults.

Exhaustive reference · Source

EvaluationResult

Single-sample evaluation paired with the observable snapshot for that sample.

Single-sample evaluation paired with the observable snapshot for that sample.

class EvaluationResult:

Requires features: python_api, python_abi, python_stubgen

sample

Kind: Getter

def sample(self) -> SampleEvaluationResult:

Evaluation record for the requested sample.

integrand_result

Kind: Getter

def integrand_result(self) -> complex:

Complex integrand value before applying the parameterization Jacobian.

integrator_weight

Kind: Getter

def integrator_weight(self) -> float:

Monte Carlo weight supplied by the integrator, excluding the parameterization Jacobian.

generated_event_count

Kind: Getter

```
def generated_event_count(self) -> Optional[int]:
```

Number of candidate events generated, or ``None`` when metadata was omitted.

accepted_event_count

Kind: Getter

```
def accepted_event_count(self) -> Optional[int]:
```

Number of generated events accepted by selectors, or ``None`` without metadata.

event_processing_time_seconds

Kind: Getter

```
def event_processing_time_seconds(self) -> Optional[float]:
```

Time spent building and selecting events, in seconds, or ``None`` without metadata.

parameterization_jacobian

Kind: Getter

```
def parameterization_jacobian(self) -> Optional[float]:
```

Sample parameterization Jacobian, or ``None`` when no Jacobian was supplied.

is_nan

Kind: Getter

```
def is_nan(self) -> Optional[bool]:
```

Whether evaluation metadata flagged a non-finite result, or ``None`` without metadata.

stability_results

Kind: Getter

```
def stability_results(self) -> Optional[list[StabilityResult]]:
```

Per-precision stability attempts, or ``None`` when metadata was omitted.

event_groups

Kind: Getter

```
def event_groups(self) -> list[EventGroup]:
```

Correlated event groups produced while evaluating this sample.

observables

Kind: Getter

```
def observables(self) -> dict:
```

Observable snapshots produced from the same Monte Carlo sample.

__str__

Kind: Method

```
def __str__(self) -> str:
```

Return a human-readable evaluation and observable summary.

__repr__

Kind: `Method`

```
def __repr__(self) -> str:
```

Return the human-readable evaluation and observable summary.

Exhaustive reference · Source

Event

Accepted cut event with momenta, its primary weight, and auxiliary weights.

Accepted cut event with momenta, its primary weight, and auxiliary weights.

```
class Event:
```

Requires features: `python_api, python_abi, python_stubgen`

incoming_momenta

Kind: `Getter`

```
def incoming_momenta(self) -> list[FourMomentum]:
```

Incoming four-momenta, aligned with ``cut_info.incoming_pdgs``.

outgoing_momenta

Kind: `Getter`

```
def outgoing_momenta(self) -> list[FourMomentum]:
```

Outgoing four-momenta, aligned with ``cut_info.outgoing_pdgs``.

cut_info

Kind: `Getter`

```
def cut_info(self) -> CutInfo:
```

Cut, graph, orientation, and loop-basis identifiers for this event.

weight

Kind: `Getter`

```
def weight(self) -> Any:
```

Primary complex event weight.

additional_weights

Kind: `Getter`

```
def additional_weights(self) -> list[AdditionalWeight]:
```

Named auxiliary complex weights such as threshold-counterterm contributions.

__str__

Kind: `Method`

```
def __str__(self) -> str:
```

Return a human-readable event summary.

__repr__

Kind: `Method`

```
def __repr__(self) -> str:
```

Return the human-readable event summary.

Exhaustive reference · Source

EventGroup

Correlated accepted events produced by one graph group for one sample.

Correlated accepted events produced by one graph group for one sample.

```
class EventGroup:
```

Requires features: `python_api, python_abi, python_stubgen`

events

Kind: `Getter`

```
def events(self) -> list[Event]:
```

Correlated accepted events in this group.

__str__

Kind: `Method`

```
def __str__(self) -> str:
```

Return a human-readable summary of the grouped events.

__repr__

Kind: `Method`

```
def __repr__(self) -> str:
```

Return the human-readable grouped-event summary.

Exhaustive reference · Source

FourMomentum

One energy-momentum four-vector returned with a generated event.

One energy-momentum four-vector returned with a generated event.

```
class FourMomentum:
```

Requires features: `python_api, python_abi, python_stubgen`

e

Kind: `Getter`

```
def e(self) -> float:
```

Energy component.

px

Kind: `Getter`

```
def px(self) -> float:
```

Momentum along the x axis.

py

Kind: `Getter`

```
def py(self) -> float:
```

Momentum along the y axis.

pz

Kind: `Getter`

```
def pz(self) -> float:
```

```
Momentum along the z axis.
```

Exhaustive reference · Source

GammaLoopAPI

```
Load, inspect, and evaluate one mutable GammaLoop session.
```

```
Load, inspect, and evaluate one mutable GammaLoop session.

One instance owns a GammaLoop state, run history, CLI settings, default runtime
settings, and session state. Calls to ``run`` and the evaluation methods all act
on that same in-memory session.

Notes
-----
``read_only_state=True`` prevents writes inside the active state directory; it
does not make this Python object immutable. Commands may still change in-memory
settings, processes, or run history. Create separate instances when independent
sessions are required.
```

```
class GammaLoopAPI:
```

Requires features: `python_api, python_abi, python_stubgen`

__new__

Kind: `AssociatedFunction`

```
def __new__(cls, state_folder: Optional[str | os.PathLike | pathlib.Path] = None,
boot_commands_path: Optional[str | os.PathLike | pathlib.Path] = None, model_file:
Optional[str | os.PathLike | pathlib.Path] = None, trace_logs_filename: Optional[str]
= None, level: Optional[LogLevel] = None, logfile_level: Optional[LogLevel] = None,
logging_prefix: object | None = None, read_only_state: bool = False,
settings_global_path: Optional[str | os.PathLike | pathlib.Path] = None,
settings_runtime_defaults_path: Optional[str | os.PathLike | pathlib.Path] = None,
clean_state: bool = False) -> GammaLoopAPI:
```

```
Load or create a GammaLoop state and initialize its CLI session.

Parameters
----------
state_folder : path-like, optional
    State directory to load or create. The default is ``./gammaloop_state``.
boot_commands_path : path-like, optional
    TOML run card whose commands are applied during startup.
model_file : path-like, optional
    Model file override used while loading or initializing the state.
trace_logs_filename : str, optional
    File receiving native trace records for this session.
level : LogLevel, optional
    Terminal log-level override for this session.
logfile_level : LogLevel, optional
    File log-level override for this session.
logging_prefix : object, optional
    Native logging-prefix configuration.
read_only_state : bool, default=False
```

```
    Prevent writes whose target lies inside the active state directory and
    disable file logging there. In-memory session changes remain possible.
settings_global_path : path-like, optional
    TOML file overriding the global settings loaded at startup.
settings_runtime_defaults_path : path-like, optional
    TOML file overriding the default runtime settings loaded at startup.
clean_state : bool, default=False
    Remove the resolved state path before startup. This is destructive and
    cannot be combined with ``read_only_state=True``.

Returns
-------
GammaLoopAPI
    A stateful API instance sharing one state, history, and settings session.

Raises
------
Exception
    If startup options conflict, the state or settings cannot be loaded, a
    boot command fails, or a boot card requests process exit.

Examples
--------
Open an existing generated state without permitting writes to it:

```python
api = GammaLoopAPI(state_folder="./state", read_only_state=True)
```
```

state_folder

Kind: Parameter

Optional[str | os.PathLike | pathlib.Path]

Default: None

State directory to load or create. The default is ``./gammaloop_state``.

boot_commands_path

Kind: Parameter

Optional[str | os.PathLike | pathlib.Path]

Default: None

TOML run card whose commands are applied during startup.

model_file

Kind: Parameter

Optional[str | os.PathLike | pathlib.Path]

Default: None

Model file override used while loading or initializing the state.

trace_logs_filename

Kind: Parameter

Optional[str]

Default: None

File receiving native trace records for this session.

level
Kind: Parameter

Optional[LogLevel]

Default: None

Terminal log-level override for this session.

logfile_level
Kind: Parameter

Optional[LogLevel]

Default: None

File log-level override for this session.

logging_prefix
Kind: Parameter

object | None

Default: None

Native logging-prefix configuration.

read_only_state
Kind: Parameter

bool

Default: False

Prevent writes whose target lies inside the active state directory and disable file logging there. In-memory session changes remain possible.

settings_global_path
Kind: Parameter

Optional[str | os.PathLike | pathlib.Path]

Default: None

TOML file overriding the global settings loaded at startup.

settings_runtime_defaults_path
Kind: Parameter

Optional[str | os.PathLike | pathlib.Path]

Default: None

TOML file overriding the default runtime settings loaded at startup.

clean_state
Kind: Parameter

bool

Default: `False`

Remove the resolved state path before startup. This is destructive and cannot be combined with ``read_only_state=True``.

evaluate_sample

Kind: `Method`

```
def evaluate_sample(self, point: Sequence[float], process_id: Optional[int] = None,
integrand_name: Optional[str] = None, use_arb_prec: bool = False, minimal_output:
bool = False, return_events: Optional[bool] = None, momentum_space: bool = False,
integrator_weight: Optional[float] = None, discrete_dim: Optional[Sequence[int]] =
None, graph_name: Optional[str] = None, orientation: Optional[int] = None) ->
EvaluationResult:
```

```
Evaluate one integration or momentum-space sample.

Parameters
----------
point : Sequence[float]
    Coordinates for one sample. In integration space, the length must match
    the selected integrand and ``discrete_dim``. In momentum space, values
    are grouped as ``(px, py, pz)`` with one triplet per independent loop
    momentum. Energy components and external momenta are not accepted.
process_id : int, optional
    Process containing the integrand. Supply this when selection is ambiguous.
integrand_name : str, optional
    Integrand to evaluate. Supply this when selection is ambiguous.
use_arb_prec : bool, default=False
    Force arbitrary-precision (Arb) internal evaluation instead of following
    the configured stability ladder.
    Returned numeric fields remain ``float64``.
minimal_output : bool, default=False
    Omit the optional evaluation metadata from the returned sample.
return_events : bool, optional
    Temporarily override event generation for this call. The integrand setting
    is restored afterward.
momentum_space : bool, default=False
    Interpret ``point`` as consecutive spatial loop-momentum ``(px, py, pz)``
    triplets instead of integration-space coordinates.
integrator_weight : float, optional
    Weight associated with this sample. The default is 1.0.
discrete_dim : Sequence[int], optional
    Discrete integration coordinates used to determine the expected dimension.
graph_name : str, optional
    Graph selected for momentum-space evaluation.
orientation : int, optional
    Orientation index for ``graph_name`` in momentum-space evaluation.

Returns
-------
EvaluationResult
    The sample evaluation and the observable snapshot for its one-row batch.

Raises
------
Exception
    If integrand selection is ambiguous or invalid, dimensions do not match,
```

```
    graph or orientation selection is invalid, or warm-up/evaluation fails.

Notes
-----
With ``use_arb_prec=False``, evaluation follows the configured ``f64``,
``f128``, and arbitrary-precision stability ladder. ``use_arb_prec=True``
forces arbitrary-precision (Arb) internal evaluation. Python-visible numeric
fields use the package's ``float64`` output contract. Evaluation may
warm the integrand and update in-memory caches or observable snapshots even in
a read-only-state session.

See Also
--------
GammaLoop's sample-evaluation contract in the interface guide and the
maintained events-and-observables example.

Examples
--------
Evaluate one point from the repository's differential API regression fixture:

```python
from pathlib import Path

from gammaloop import GammaLoopAPI

example = Path("examples/api/python/epem_a_ddxg_xs_LO")
api = GammaLoopAPI(
    state_folder=example / "state",
    boot_commands_path=example / "run.toml",
    clean_state=True,
)
point = [0.17, 0.31, 0.53, 0.23, 0.41, 0.67]
result = api.evaluate_sample(point, return_events=True)
assert result.parameterization_jacobian is not None
assert result.stability_results
assert result.event_groups
```

This card verifies API and event plumbing. Its powered coupling selector is not an
independently reviewed perturbative-order definition or normalization benchmark.
```

point

Kind: Parameter

Sequence[float]

Coordinates for one sample. In integration space, the length must match the selected integrand and ``discrete_dim``. In momentum space, values are grouped as ``(px, py, pz)`` with one triplet per independent loop momentum. Energy components and external momenta are not accepted.

process_id

Kind: Parameter

Optional[int]

Default: None

Process containing the integrand. Supply this when selection is ambiguous.

integrand_name

Kind: Parameter

Optional[str]

Default: None

Integrand to evaluate. Supply this when selection is ambiguous.

use_arb_prec

Kind: Parameter

bool

Default: False

Force arbitrary-precision (Arb) internal evaluation instead of following the configured stability ladder. Returned numeric fields remain ``float64``.

minimal_output

Kind: Parameter

bool

Default: False

Omit the optional evaluation metadata from the returned sample.

return_events

Kind: Parameter

Optional[bool]

Default: None

Temporarily override event generation for this call. The integrand setting is restored afterward.

momentum_space

Kind: Parameter

bool

Default: False

Interpret ``point`` as consecutive spatial loop-momentum ``(px, py, pz)`` triplets instead of integration-space coordinates.

integrator_weight

Kind: Parameter

Optional[float]

Default: None

Weight associated with this sample. The default is 1.0.

discrete_dim

Kind: Parameter

Optional[Sequence[int]]

Default: None

Discrete integration coordinates used to determine the expected dimension.

graph_name

Kind: Parameter

Optional[str]

Default: None

Graph selected for momentum-space evaluation.

orientation

Kind: Parameter

Optional[int]

Default: None

Orientation index for ``graph_name`` in momentum-space evaluation.

evaluate_samples

Kind: Method

```
def evaluate_samples(self, points: numpy.NDArray[numpy.float64], process_id:
Optional[int] = None, integrand_name: Optional[str] = None, use_arb_prec: bool =
False, minimal_output: bool = False, return_events: Optional[bool] = None,
momentum_space: bool = False, integrator_weights:
Optional[numpy.NDArray[numpy.float64]] = None, discrete_dims:
numpy.NDArray[numpy.unsignedinteger] | None = None, graph_names:
Optional[Sequence[Optional[str]]] = None, orientations:
Optional[Sequence[Optional[int]]] = None) -> BatchEvaluationResult:
```

Evaluate a batch of integration or momentum-space samples.

```
Parameters
----------
points : numpy.ndarray[numpy.float64]
    Two-dimensional array with one sample per row. Integration-space columns
    must match the selected integrand; momentum-space columns are flattened
    ``(px, py, pz)`` groups with one triplet per independent loop momentum.
    Energy components and external momenta are not accepted.
process_id : int, optional
    Process containing the integrand. Supply this when selection is ambiguous.
integrand_name : str, optional
    Integrand to evaluate. Supply this when selection is ambiguous.
use_arb_prec : bool, default=False
    Force arbitrary-precision (Arb) internal evaluation instead of following
    the configured stability ladder.
    Returned numeric fields remain ``float64``.
minimal_output : bool, default=False
    Omit the optional evaluation metadata from every returned sample.
return_events : bool, optional
    Temporarily override event generation for this call. The integrand setting
    is restored afterward.
momentum_space : bool, default=False
    Interpret each row as consecutive spatial loop-momentum ``(px, py, pz)``
    triplets instead of integration-space coordinates.
integrator_weights : numpy.ndarray[numpy.float64], optional
```

```
    One weight per row. Defaults to 1.0 for every sample.
discrete_dims : numpy.ndarray[numpy.unsignedinteger], optional
    Two-dimensional array with one row of discrete coordinates per sample.
graph_names : Sequence[str | None], optional
    Momentum-space graph selection for each sample.
orientations : Sequence[int | None], optional
    Momentum-space orientation selection for each sample.

Returns
-------
BatchEvaluationResult
    Per-sample evaluations and one observable snapshot for the complete batch.

Raises
------
Exception
    If integrand selection is ambiguous or invalid, array or option lengths do
    not match, dimensions are invalid, or warm-up/evaluation fails.

Notes
-----
With ``use_arb_prec=False``, evaluation follows the configured ``f64``,
``f128``, and arbitrary-precision stability ladder. ``use_arb_prec=True``
forces arbitrary-precision (Arb) internal evaluation. Python-visible numeric
fields use the package's ``float64`` output contract. Evaluation may
update in-memory caches or observable snapshots in a read-only-state session.

See Also
--------
GammaLoop's sample-evaluation contract in the interface guide and the
maintained events-and-observables example.

Examples
--------
Evaluate two rows and inspect their per-sample events and batch-level histograms:

```python
from pathlib import Path

import numpy as np
from gammaloop import GammaLoopAPI

example = Path("examples/api/python/epem_a_ddxg_xs_LO")
api = GammaLoopAPI(
    state_folder=example / "state",
    boot_commands_path=example / "run.toml",
    clean_state=True,
)
points = np.array([
    [0.17, 0.31, 0.53, 0.23, 0.41, 0.67],
    [0.11, 0.29, 0.47, 0.19, 0.37, 0.59],
], dtype=float)
result = api.evaluate_samples(points, return_events=True)
assert len(result.samples) == 2
assert all(sample.event_groups for sample in result.samples)
assert result.observables["leading_jet_pt_hist"].sample_count == 2
```
```

```
assert len(result.observables["leading_jet_pt_hist"].bins) == 8
```

The fixture exercises the API surface; its powered coupling selector is not a validated
perturbative-order or normalization benchmark.

points

Kind: Parameter

numpy.NDArray[numpy.float64]

Two-dimensional array with one sample per row. Integration-space columns must match the selected integrand; momentum-space columns are flattened ``(px, py, pz)`` groups with one triplet per independent loop momentum. Energy components and external momenta are not accepted.

process_id

Kind: Parameter

Optional[int]

Default: None

Process containing the integrand. Supply this when selection is ambiguous.

integrand_name

Kind: Parameter

Optional[str]

Default: None

Integrand to evaluate. Supply this when selection is ambiguous.

use_arb_prec

Kind: Parameter

bool

Default: False

Force arbitrary-precision (Arb) internal evaluation instead of following the configured stability ladder. Returned numeric fields remain ``float64``.

minimal_output

Kind: Parameter

bool

Default: False

Omit the optional evaluation metadata from every returned sample.

return_events

Kind: Parameter

Optional[bool]

Default: None

Temporarily override event generation for this call. The integrand setting is restored afterward.

momentum_space

Kind: Parameter

bool

Default: False

Interpret each row as consecutive spatial loop-momentum ``(px, py, pz)`` triplets instead of integration-space coordinates.

integrator_weights

Kind: Parameter

Optional[numpy.NDArray[numpy.float64]]

Default: None

One weight per row. Defaults to 1.0 for every sample.

discrete_dims

Kind: Parameter

numpy.NDArray[numpy.unsignedinteger] | None

Default: None

Two-dimensional array with one row of discrete coordinates per sample.

graph_names

Kind: Parameter

Optional[Sequence[Optional[str]]]

Default: None

Momentum-space graph selection for each sample.

orientations

Kind: Parameter

Optional[Sequence[Optional[int]]]

Default: None

Momentum-space orientation selection for each sample.

import_graphs

Kind: Method

```
def import_graphs(self, graphs: str, process_name: Optional[str] = None, process_id:
Optional[int] = None, integrand_name: Optional[str] = None, format: str = 'dot',
overwrite: bool = False, append: bool = False) -> None:
```

Import DOT graphs into a new or existing process/integrand collection.

```
Parameters
----------
graphs : str
    DOT file path when ``format="dot"`` or inline DOT text when
    ``format="string"``.
process_name, process_id : str or int, optional
    Process to create or update. Inline text requires one of these selectors.
integrand_name : str, optional
```

```
    Integrand within the selected process.
format : {"dot", "string"}, default="dot"
    Select file-backed or inline input.
overwrite, append : bool, default=False
    Replace an existing collection or append to it; these modes conflict.

Raises
------
Exception
    If selectors conflict, the source is missing or malformed, or importing fails.
```

graphs

Kind: Parameter

str

DOT file path when ``format="dot"`` or inline DOT text when ``format="string"``.

process_name

Kind: Parameter

Optional[str]

Default: None

Process to create or update. Inline text requires one of these selectors.

process_id

Kind: Parameter

Optional[int]

Default: None

Process to create or update. Inline text requires one of these selectors.

integrand_name

Kind: Parameter

Optional[str]

Default: None

Integrand within the selected process.

format

Kind: Parameter

str

Default: 'dot'

Select file-backed or inline input.

overwrite

Kind: Parameter

bool

Default: False

Replace an existing collection or append to it; these modes conflict.

append

Kind: Parameter

bool

Default: False

Replace an existing collection or append to it; these modes conflict.

get_lmbs

Kind: Method

```
def get_lmbs(self, graphs: str, format: str = 'dot') -> list[list[tuple[list[int],
list[int], dict[int, tuple[list[int], list[int]]]]]]:
```

Generate loop-momentum bases for each supplied graph.

```
Parameters
----------
graphs : str
    DOT file path or inline DOT text, as selected by ``format``.
format : {"dot", "string"}, default="dot"
    Select file-backed or inline input.

Returns
-------
list
    Per graph, a list of ``(loop_edges, external_edges, edge_signatures)``
    tuples. Each signature contains its loop and external coefficients.
```

graphs

Kind: Parameter

str

DOT file path or inline DOT text, as selected by ``format``.

format

Kind: Parameter

str

Default: 'dot'

Select file-backed or inline input.

get_orientations

Kind: Method

```
def get_orientations(self, graph_name: str, process_id: Optional[int] = None,
integrand_name: Optional[str] = None) -> list[dict[int, int]]:
```

Return the causal-flow orientations generated for one graph.

```
Each returned dictionary maps an edge id to ``1`` (default), ``-1``
(reversed), or ``0`` (undirected). Supply process and integrand selectors when
the active state does not identify a unique integrand.

Parameters
----------
graph_name : str
```

```
    Name of the graph within the selected integrand.
process_id : int, optional
    Numeric process identifier; omit when process selection is unambiguous.
integrand_name : str, optional
    Integrand containing the graph; omit when integrand selection is unambiguous.

Returns
-------
list[dict[int, int]]
    One edge-direction mapping per generated orientation.
```

graph_name

Kind: Parameter

```
str
```

Name of the graph within the selected integrand.

process_id

Kind: Parameter

```
Optional[int]
```

Default: None

Numeric process identifier; omit when process selection is unambiguous.

integrand_name

Kind: Parameter

```
Optional[str]
```

Default: None

Integrand containing the graph; omit when integrand selection is unambiguous.

get_model

Kind: Method

```
def get_model(self) -> str:
```

Serialize the active physics model as JSON.

```
Returns
-------
str
    JSON representation of the model currently owned by this session.
```

evaluate

Kind: Method

```
def evaluate(self, process_id: Optional[int] = None, graphs_group_name: Optional[str]
= None, result_path: Optional[str | os.PathLike | pathlib.Path] = None, numerical:
bool = True, number_of_terms_in_epsilon_expansion: Optional[int] = None) -> str:
```

Evaluate one generated graph group as a symbolic or numerical expression.

```
Parameters
----------
process_id : int, optional
    Process containing the requested graph group.
```

```
graphs_group_name : str, optional
    Group to evaluate when the current state is ambiguous.
result_path : path-like, optional
    Optional destination for the evaluated expression.
numerical : bool, default=True
    Evaluate numerically instead of retaining a symbolic result.
number_of_terms_in_epsilon_expansion : int, optional
    Truncate the dimensional-regulator expansion to this many terms.

Returns
-------
str
    Canonical Symbolica representation of the result.
```

process_id

Kind: Parameter

`Optional[int]`

Default: `None`

Process containing the requested graph group.

graphs_group_name

Kind: Parameter

`Optional[str]`

Default: `None`

Group to evaluate when the current state is ambiguous.

result_path

Kind: Parameter

`Optional[str | os.PathLike | pathlib.Path]`

Default: `None`

Optional destination for the evaluated expression.

numerical

Kind: Parameter

`bool`

Default: `True`

Evaluate numerically instead of retaining a symbolic result.

number_of_terms_in_epsilon_expansion

Kind: Parameter

`Optional[int]`

Default: `None`

Truncate the dimensional-regulator expansion to this many terms.

import_model

Kind: Method

```
def import_model(self, model_specifier: str | os.PathLike | pathlib.Path,
simplify_model: bool = True) -> None:
```

Replace the active model from a GammaLoop model file or supported model source.

``simplify_model`` applies the standard symbolic simplification pass while importing. Existing process data may no longer be compatible with a replaced model, so import the model before generating processes.

```
Parameters
----------
model_specifier : path-like
    Path or model specifier accepted by GammaLoop's model importer.
simplify_model : bool, default=True
    Apply the standard symbolic simplification pass while importing.
```

model_specifier

Kind: Parameter

```
str | os.PathLike | pathlib.Path
```

Path or model specifier accepted by GammaLoop's model importer.

simplify_model

Kind: Parameter

```
bool
```

Default: True

Apply the standard symbolic simplification pass while importing.

list_outputs

Kind: Method

```
def list_outputs(self) -> tuple[dict[str, int], dict[str, int]]:
```

List generated amplitude and cross-section names with their process ids.

```
Returns
-------
tuple[dict[str, int], dict[str, int]]
    Amplitude mapping followed by the cross-section mapping.
```

get_integrand_info

Kind: Method

```
def get_integrand_info(self, process_id: Optional[int] = None, integrand_name:
Optional[str] = None) -> IntegrandInfo:
```

Describe the selected generated integrand and its graph structure.

```
Parameters
----------
process_id : int, optional
    Process containing the integrand. Supply this when selection is ambiguous.
integrand_name : str, optional
    Integrand to inspect. Supply this when selection is ambiguous.

Returns
-------
```

```
IntegrandInfo
    Structured process, backend, graph, orientation, cut, and size metadata.

Raises
------
Exception
    If no unique generated integrand matches the selection.

Examples
--------
Inspect the generated graph groups in the repository's differential API fixture:

```python
from pathlib import Path

from gammaloop import GammaLoopAPI

example = Path("examples/api/python/epem_a_ddxg_xs_LO")
api = GammaLoopAPI(
    state_folder=example / "state",
    boot_commands_path=example / "run.toml",
    clean_state=True,
)
info = api.get_integrand_info()
assert info.kind == "cross section"
assert info.graph_count == 2
assert info.graph_group_count == len(info.graph_groups)
assert all(
    sum(graph.is_master for graph in group.graphs) == 1
    for group in info.graph_groups
)
```

This fixture's powered coupling selector is a regression input, not a reviewed
physical
perturbative-order definition.
```

process_id

Kind: Parameter

Optional[int]

Default: None

Process containing the integrand. Supply this when selection is ambiguous.

integrand_name

Kind: Parameter

Optional[str]

Default: None

Integrand to inspect. Supply this when selection is ambiguous.

get_integrand_settings

Kind: Method

```
def get_integrand_settings(self, process_id: Optional[int] = None, integrand_name:
Optional[str] = None) -> SettingsValue:
```

Return a detached, read-only snapshot of one integrand's settings.

```
Parameters
----------
process_id : int, optional
    Process containing the integrand. Supply this when selection is ambiguous.
integrand_name : str, optional
    Integrand whose settings are required.

Returns
-------
SettingsValue
    Serialized settings snapshot supporting ``get(path)``, attribute access,
    indexing, and ``to_dict()``. Mutating it does not update the live session.

Raises
------
Exception
    If selection fails, the integrand has not been generated, or its settings
    cannot be serialized.

Examples
--------
Read a nested setting without changing the live integrand:

```python
settings = api.get_integrand_settings(process_id=0)
print(settings.get("general.generate_events"))
```
```

process_id

Kind: Parameter

Optional[int]

Default: None

Process containing the integrand. Supply this when selection is ambiguous.

integrand_name

Kind: Parameter

Optional[str]

Default: None

Integrand whose settings are required.

get_run_history

Kind: Method

```
def get_run_history(self) -> str:
```

Render the current in-memory run history as TOML.

```
Returns
-------
```

```
str
    TOML representation of the history owned by this API instance.

Raises
------
Exception
    If the current history cannot be serialized.

Notes
-----
This reports the live session, including commands run through ``run``; it does
not imply that the history has been persisted to the state directory.

Examples
--------
Capture a reproducible run card from the current session:

```python
run_card_toml = api.get_run_history()
```
```

get_global_settings

Kind: Method

```
def get_global_settings(self) -> str:

Render the current effective CLI and global settings as TOML.

Returns
-------
str
    TOML representation of the settings used by this API session.

Raises
------
Exception
    If the settings cannot be rendered.

Examples
--------
Record the effective settings after applying startup overrides:

```python
settings_toml = api.get_global_settings()
```
```

get_active_command_blocks

Kind: Method

```
def get_active_command_blocks(self) -> dict[str, list[str]]:

Return the named command blocks in the current run history.

Returns
-------
dict[str, list[str]]
    Mapping from block name to its rendered CLI commands.
```

Examples

Inspect the commands currently grouped under each block:

```python
for name, commands in api.get_active_command_blocks().items():
    print(name, commands)
```

get_default_runtime_settings

Kind: Method

```
def get_default_runtime_settings(self) -> SettingsValue:
```

Return a detached, read-only snapshot of the default runtime settings.

Returns

SettingsValue
    Serialized settings including defaults. Use ``get(path)`` or ``to_dict()``
    to inspect it; changes to derived Python values do not affect the session.

Raises

Exception
    If the settings cannot be serialized.

Examples

Inspect a runtime default by its documented settings path:

```python
runtime = api.get_default_runtime_settings()
print(runtime.get("integrator.n_start"))
```

get_dot_files

Kind: Method

```
def get_dot_files(self, process: Optional[int | str] = None, integrand_name:
Optional[str] = None, settings: DotExportSettings = ...) -> str:
```

Render a selected amplitude or cross section as Graphviz DOT text.

Parameters
----
process : int or str, optional
    Process id or name; omit only when selection is unambiguous.
integrand_name : str, optional
    Integrand to render.
settings : DotExportSettings, optional
    Controls diagram combination, UV terms, algebra, and generated fields.

Returns
----
str
    DOT source suitable for Graphviz or GammaLoop's drawing pipeline.

process

Kind: Parameter

Optional[int | str]

Default: None

Process id or name; omit only when selection is unambiguous.

integrand_name

Kind: Parameter

Optional[str]

Default: None

Integrand to render.

settings

Kind: Parameter

DotExportSettings

Default: ...

Controls diagram combination, UV terms, algebra, and generated fields.

run

Kind: Method

```
def run(self, command: str) -> None:
```

```
Parse and execute a CLI command in this API instance's session.

Parameters
----------
command : str
    GammaLoop CLI command text.

Returns
-------
None
    The command's normal output is emitted through the configured CLI/logging
    sinks; inspect structured state through the corresponding getter methods.

Raises
------
Exception
    If the command cannot be parsed or execution fails.

Notes
-----
Commands share and may mutate the instance's in-memory state, settings, and run
history. ``read_only_state=True`` only blocks writes inside the active state
directory. The API does not automatically persist the session after this call;
persistence depends on the executed command's explicit output behavior.

Examples
--------
Display the processes loaded in the current state:
```

```python
api.run("display processes")
```

command

Kind: Parameter

str

GammaLoop CLI command text.

generate_cff

Kind: Method

```
def generate_cff(self, dot_string: str, subgraph_nodes: Sequence[str],
reverse_dangling: Sequence[int], orientation_pattern: Optional[str] = None) ->
list[tuple[dict[int, int], str]]:
```

Build a causal-flow expression from an inline DOT graph or one of its subgraphs.

```
Parameters
----------
dot_string : str
    Inline DOT graph using particles from the active model.
subgraph_nodes : Sequence[str]
    Vertex names retained in the subgraph; an empty sequence selects all nodes.
reverse_dangling : Sequence[int]
    Dangling edge ids whose orientation is reversed.
orientation_pattern : str, optional
    Pattern restricting returned causal-flow orientations.

Returns
-------
list[tuple[dict[int, int], str]]
    Edge-direction maps paired with their energy-denominator expressions.
```

dot_string

Kind: Parameter

str

Inline DOT graph using particles from the active model.

subgraph_nodes

Kind: Parameter

Sequence[str]

Vertex names retained in the subgraph; an empty sequence selects all nodes.

reverse_dangling

Kind: Parameter

Sequence[int]

Dangling edge ids whose orientation is reversed.

orientation_pattern

Kind: Parameter

Optional[str]

Default: None

Pattern restricting returned causal-flow orientations.

generate_cff_as_json_string

Kind: Method

def generate_cff_as_json_string(self, dot_string: str, subgraph_nodes: Sequence[str], reverse_dangling: Sequence[int], orientation_pattern: Optional[str] = None) -> str:

Serialize a causal-flow expression and its surfaces as JSON.

This accepts the same graph, subgraph, and dangling-edge inputs as
``generate_cff``. The current JSON representation is intended for GammaLoop
tooling and may contain internal structural details.

Parameters
----
dot_string : str
    Inline DOT graph using particles from the active model.
subgraph_nodes : Sequence[str]
    Vertex names retained in the subgraph; an empty sequence selects all nodes.
reverse_dangling : Sequence[int]
    Dangling edge ids whose orientation is reversed.
orientation_pattern : str, optional
    Pattern restricting returned causal-flow orientations.

Returns
----
str
    JSON representation of the causal-flow expression and E-surfaces.

dot_string

Kind: Parameter

str

Inline DOT graph using particles from the active model.

subgraph_nodes

Kind: Parameter

Sequence[str]

Vertex names retained in the subgraph; an empty sequence selects all nodes.

reverse_dangling

Kind: Parameter

Sequence[int]

Dangling edge ids whose orientation is reversed.

orientation_pattern

Kind: Parameter

Optional[str]

Default: None

Pattern restricting returned causal-flow orientations.

Exhaustive reference · Source

HistogramAccumulator

Mutable continuous or discrete histogram accumulator with sample-level statistics.

Mutable continuous or discrete histogram accumulator with sample-level statistics.

Notes

The example below deliberately places at most one entry in each bin. If several correlated
entries land in one bin, the current helper records their squared weights separately rather
than grouping their weights like GammaLoop's native observable pipeline. Do not replay raw
event groups through this class until that statistical contract is aligned.

Examples

Merge two pending, statistically independent samples before committing them:

```python
from gammaloop import HistogramAccumulator

left = HistogramAccumulator.continuous("energy", 0.0, 4.0, 4)
right = HistogramAccumulator.continuous("energy", 0.0, 4.0, 4)
left.fill_continuous_sample([(0.5, 2.0)])
right.fill_continuous_sample([(2.5, 3.0)])
left.merge_in_place(right)
left.update_results()

snapshot = left.snapshot()
assert snapshot.sample_count == 2
assert snapshot.bins[0].sum_weights == 2.0
assert snapshot.bins[0].sum_weights_squared == 4.0
assert snapshot.bins[0].sum_weights / snapshot.sample_count == 1.0
assert snapshot.bins[2].sum_weights == 3.0
assert snapshot.bins[2].sum_weights_squared == 9.0
assert right.snapshot().sample_count == 0
assert len(left.rebin(2).snapshot().bins) == 2
```

class HistogramAccumulator:

Requires features: `python_api, python_abi, python_stubgen`

continuous

Kind: `AssociatedFunction`

def continuous(title: str, x_min: float, x_max: float, n_bins: int, type_description: str = 'AL', phase: str = 'real', value_transform: str = 'identity', log_x_axis: bool = False, log_y_axis: bool = True) -> HistogramAccumulator:

Create an evenly binned continuous histogram accumulator.

Parameters

```
title : str
    Human-readable title used by snapshots and exported output.
x_min : float
    Lower edge of the histogram range.
x_max : float
    Upper edge of the histogram range.
n_bins : int
    Number of equal-width in-range bins.
type_description : str, default="AL"
    HwU ``TYPE@`` metadata.
phase : {"real", "imag"}, default="real"
    Component of each complex event weight to accumulate.
value_transform : {"identity", "log10"}, default="identity"
    Transformation applied to coordinates before binning.
log_x_axis : bool, default=False
    Request logarithmic horizontal-axis rendering.
log_y_axis : bool, default=True
    Request logarithmic vertical-axis rendering.
```

title

Kind: Parameter

str

Human-readable title used by snapshots and exported output.

x_min

Kind: Parameter

float

Lower edge of the histogram range.

x_max

Kind: Parameter

float

Upper edge of the histogram range.

n_bins

Kind: Parameter

int

Number of equal-width in-range bins.

type_description

Kind: Parameter

str

Default: 'AL'

HwU ``TYPE@`` metadata.

phase

Kind: Parameter

str

Default: 'real'

Component of each complex event weight to accumulate.

value_transform

Kind: Parameter

str

Default: 'identity'

Transformation applied to coordinates before binning.

log_x_axis

Kind: Parameter

bool

Default: False

Request logarithmic horizontal-axis rendering.

log_y_axis

Kind: Parameter

bool

Default: True

Request logarithmic vertical-axis rendering.

discrete

Kind: AssociatedFunction

```
def discrete(title: str, min_bin_id: int, max_bin_id: int, ordering: str =
'ascending_bin_id', labels: Optional[Sequence[str]] = None, type_description: str =
'AL', phase: str = 'real', log_y_axis: bool = True) -> HistogramAccumulator:
```

Create a discrete histogram accumulator over an inclusive integer range.

```
Parameters
----------
title : str
    Human-readable title used by snapshots and exported output.
min_bin_id : int
    Smallest accepted bin identifier.
max_bin_id : int
    Largest accepted bin identifier, inclusive.
ordering : {"ascending_bin_id", "value_descending", "abs_value_descending"},
default="ascending_bin_id"
    Ordering used when bins are returned or exported.
labels : Sequence[str], optional
    One label per bin, ordered by ascending bin identifier.
type_description : str, default="AL"
    HwU ``TYPE@`` metadata.
phase : {"real", "imag"}, default="real"
    Component of each complex event weight to accumulate.
log_y_axis : bool, default=True
    Request logarithmic vertical-axis rendering.

Raises
------
```

ValueError
    If the range, label count, ordering, or phase is invalid.

title
Kind: Parameter

str

Human-readable title used by snapshots and exported output.

min_bin_id
Kind: Parameter

int

Smallest accepted bin identifier.

max_bin_id
Kind: Parameter

int

Largest accepted bin identifier, inclusive.

ordering
Kind: Parameter

str

Default: 'ascending_bin_id'

Ordering used when bins are returned or exported.

labels
Kind: Parameter

Optional[Sequence[str]]

Default: None

One label per bin, ordered by ascending bin identifier.

type_description
Kind: Parameter

str

Default: 'AL'

HwU ``TYPE@`` metadata.

phase
Kind: Parameter

str

Default: 'real'

Component of each complex event weight to accumulate.

log_y_axis
Kind: Parameter

bool

Default: `True`

```
Request logarithmic vertical-axis rendering.
```

snapshot

Kind: Method

```
def snapshot(self) -> HistogramSnapshot:
```

```
Return an immutable snapshot including committed and pending samples.
```

merge_in_place

Kind: Method

```
def merge_in_place(self, other: HistogramAccumulator) -> None:
```

```
Move pending samples from another compatible accumulator into this one.

Parameters
----------
other : HistogramAccumulator
    Compatible accumulator whose pending samples are consumed.

Raises
------
ValueError
    If histogram definitions differ.
```

other

Kind: Parameter

```
HistogramAccumulator
```

```
Compatible accumulator whose pending samples are consumed.
```

rebin

Kind: Method

```
def rebin(self, contiguous_bins: int) -> HistogramAccumulator:
```

```
Return a copy with each run of adjacent continuous bins combined.

Discrete histograms are copied unchanged.

Parameters
----------
contiguous_bins : int
    Positive number of old bins per new bin; it must divide the bin count.
```

contiguous_bins

Kind: Parameter

```
int
```

```
Positive number of old bins per new bin; it must divide the bin count.
```

rescale

Kind: Method

```
def rescale(self, factor: float) -> None:
```

```
Multiply all accumulated weights by a constant in place.

Parameters
----------
factor : float
    Scale applied to weight sums; squared-weight sums use its square.
```

factor

Kind: Parameter

```
float
```

```
Scale applied to weight sums; squared-weight sums use its square.
```

change_bin_ordering

Kind: Method

```
def change_bin_ordering(self, ordering: str) -> None:
```

```
Change the display ordering of a discrete histogram.

Parameters
----------
ordering : {"ascending_bin_id", "value_descending", "abs_value_descending"}
    New ordering for snapshots and exported output.

Raises
------
ValueError
    If this is a continuous histogram or the name is invalid.
```

ordering

Kind: Parameter

```
str
```

```
New ordering for snapshots and exported output.
```

update_results

Kind: Method

```
def update_results(self) -> None:
```

```
Commit pending samples to the accumulator's completed-result counters.
```

fill_continuous_sample

Kind: Method

```
def fill_continuous_sample(self, entries: Sequence[tuple[float, float]]) -> None:
```

```
Add one independent continuous sample containing coordinate-weight pairs.

Parameters
----------
entries : Sequence[tuple[float, float]]
    ``(coordinate, projected_weight)`` entries for this sample.

Raises
------
ValueError
    If this is not a continuous histogram.
```

entries

Kind: Parameter

Sequence[tuple[float, float]]

``(coordinate, projected_weight)`` entries for this sample.

fill_discrete_sample

Kind: Method

def fill_discrete_sample(self, entries: Sequence[tuple[int, float]]) -> None:

Add one independent discrete sample containing bin-weight pairs.

Parameters
----
entries : Sequence[tuple[int, float]]
    ``(bin_id, projected_weight)`` entries for this sample.

Raises

ValueError
    If this is not a discrete histogram.

entries

Kind: Parameter

Sequence[tuple[int, float]]

``(bin_id, projected_weight)`` entries for this sample.

Exhaustive reference · Source

HistogramBin

Raw statistics and optional coordinate metadata for one histogram bin.

Raw statistics and optional coordinate metadata for one histogram bin.

class HistogramBin:

Requires features: python_api, python_abi, python_stubgen

x_min

Kind: Getter

def x_min(self) -> Optional[float]:

Lower coordinate edge, or ``None`` for a discrete or overflow bin.

x_max

Kind: Getter

def x_max(self) -> Optional[float]:

Upper coordinate edge, or ``None`` for a discrete or underflow bin.

bin_id

Kind: Getter

def bin_id(self) -> Optional[int]:

Discrete bin identifier, or ``None`` for a continuous bin.

label

Kind: Getter

```
def label(self) -> Optional[str]:
```

Optional label of a discrete bin.

entry_count

Kind: Getter

```
def entry_count(self) -> int:
```

Number of event entries accumulated in this bin.

sum_weights

Kind: Getter

```
def sum_weights(self) -> float:
```

Sum of per-sample projected weights for this bin.

sum_weights_squared

Kind: Getter

```
def sum_weights_squared(self) -> float:
```

Sum of squared per-sample projected weights for this bin.

mitigated_fill_count

Kind: Getter

```
def mitigated_fill_count(self) -> int:
```

Number of fills produced by paired boundary-misbinning mitigation.

average

Kind: Method

```
def average(self, sample_count: int) -> float:
```

Compute this bin's Monte Carlo mean from its raw weight sum.

```
Parameters
----------
sample_count : int
    Number of statistically independent samples represented by the histogram.

Returns
-------
float
    ``sum_weights / sample_count``, or zero when ``sample_count`` is zero.
```

sample_count

Kind: Parameter

```
int
```

Number of statistically independent samples represented by the histogram.

error

Kind: Method

```
def error(self, sample_count: int) -> float:
```

```
Compute the standard error of this bin's Monte Carlo mean.

Parameters
----------
sample_count : int
    Number of statistically independent samples represented by the histogram.

Returns
-------
float
    Sample-mean standard error, or zero when it cannot be estimated reliably.
```

sample_count

Kind: Parameter

int

Number of statistically independent samples represented by the histogram.

Exhaustive reference · Source

HistogramSnapshot

Immutable, mergeable histogram snapshot containing raw per-bin statistics.

Immutable, mergeable histogram snapshot containing raw per-bin statistics.

class HistogramSnapshot:

Requires features: python_api, python_abi, python_stubgen

kind

Kind: Getter

def kind(self) -> str:

``"continuous"`` or ``"discrete"``.

title

Kind: Getter

def title(self) -> str:

Human-readable histogram title.

type_description

Kind: Getter

def type_description(self) -> str:

HwU ``TYPE@`` description retained with the histogram.

phase

Kind: Getter

def phase(self) -> str:

Complex-weight component accumulated: ``"real"`` or ``"imag"``.

value_transform

Kind: Getter

def value_transform(self) -> str:

Coordinate transform: ``"identity"`` or ``"log10"``.

supports_misbinning_mitigation
Kind: Getter

def supports_misbinning_mitigation(self) -> bool:

Whether paired fills can mitigate bin-boundary instability.

x_min
Kind: Getter

def x_min(self) -> Optional[float]:

Lower bound of a continuous histogram, otherwise ``None``.

x_max
Kind: Getter

def x_max(self) -> Optional[float]:

Upper bound of a continuous histogram, otherwise ``None``.

sample_count
Kind: Getter

def sample_count(self) -> int:

Number of statistically independent Monte Carlo samples represented.

log_x_axis
Kind: Getter

def log_x_axis(self) -> bool:

Whether compatible renderers should use a logarithmic horizontal axis.

log_y_axis
Kind: Getter

def log_y_axis(self) -> bool:

Whether compatible renderers should use a logarithmic vertical axis.

discrete_min_bin_id
Kind: Getter

def discrete_min_bin_id(self) -> Optional[int]:

Smallest discrete bin identifier, otherwise ``None``.

discrete_ordering
Kind: Getter

def discrete_ordering(self) -> Optional[str]:

Discrete display ordering, otherwise ``None``.

bins
Kind: Getter

def bins(self) -> list[HistogramBin]:

Raw statistics for every in-range bin, in display order.

underflow_bin
Kind: Getter

```
def underflow_bin(self) -> HistogramBin:
```

Raw statistics for entries below a continuous histogram's range.

overflow_bin
Kind: Getter

```
def overflow_bin(self) -> HistogramBin:
```

Raw statistics for entries above a continuous histogram's range.

statistics
Kind: Getter

```
def statistics(self) -> HistogramStats:
```

Histogram-wide entry and mitigation counters.

Exhaustive reference · Source

HistogramStats

Aggregate entry, NaN, and misbinning-mitigation counts for a histogram.

Aggregate entry, NaN, and misbinning-mitigation counts for a histogram.

```
class HistogramStats:
```

Requires features: `python_api, python_abi, python_stubgen`

in_range_entry_count
Kind: Getter

```
def in_range_entry_count(self) -> int:
```

Number of event entries assigned to in-range bins.

nan_value_count
Kind: Getter

```
def nan_value_count(self) -> int:
```

Number of entries skipped because their coordinate was non-finite.

mitigated_pair_count
Kind: Getter

```
def mitigated_pair_count(self) -> int:
```

Number of entry pairs processed by boundary-misbinning mitigation.

Exhaustive reference · Source

IntegrandActiveThresholdCut

Viable cut associated with a threshold E-surface.

Viable cut associated with a threshold E-surface.

```
class IntegrandActiveThresholdCut:
```

Requires features: `python_api, python_abi, python_stubgen`

cut_id
Kind: Getter

```
def cut_id(self) -> int:
```

Zero-based identifier of a cut on which the E-surface remains viable.

can_become_pinched
Kind: Getter

def can_become_pinched(self) -> bool:

Whether this cut can make the E-surface pinched for some kinematics.

Exhaustive reference · Source

IntegrandCut
Cut edges, raising power, and threshold associations for one cross-section cut.

Cut edges, raising power, and threshold associations for one cross-section cut.

class IntegrandCut:

Requires features: python_api, python_abi, python_stubgen

cut_id
Kind: Getter

def cut_id(self) -> int:

Zero-based cut identifier within the representative graph.

edge_ids
Kind: Getter

def edge_ids(self) -> list[int]:

Edge identifiers crossed by the cut.

raising_power
Kind: Getter

def raising_power(self) -> int:

Maximum multiplicity of the cut's related threshold group.

left_thresholds
Kind: Getter

def left_thresholds(self) -> list[IntegrandCutThreshold]:

Threshold associations on the left side of the cut.

right_thresholds
Kind: Getter

def right_thresholds(self) -> list[IntegrandCutThreshold]:

Threshold associations on the right side of the cut.

Exhaustive reference · Source

IntegrandCutThreshold
Classification and boundary edges of one threshold associated with a cut.

Classification and boundary edges of one threshold associated with a cut.

class IntegrandCutThreshold:

Requires features: python_api, python_abi, python_stubgen

esurface_id
Kind: Getter

```
def esurface_id(self) -> int:
```

Threshold E-surface identifier within the graph group.

status
Kind: Getter

```
def status(self) -> str:
```

Viability classification for this threshold-cut association.

cut_boundary_edge_ids
Kind: Getter

```
def cut_boundary_edge_ids(self) -> list[int]:
```

Cut edges that bound the associated threshold region.

threshold_boundary_edge_ids
Kind: Getter

```
def threshold_boundary_edge_ids(self) -> list[int]:
```

Threshold edges that bound the associated threshold region.

invariant_bound_is_applicable
Kind: Getter

```
def invariant_bound_is_applicable(self) -> bool:
```

Whether the invariant-bound criterion applies to this association.

Exhaustive reference · Source

IntegrandGraph
Identity and master-graph status of one graph in an integrand.

Identity and master-graph status of one graph in an integrand.

```
class IntegrandGraph:
```

Requires features: `python_api, python_abi, python_stubgen`

graph_id
Kind: Getter

```
def graph_id(self) -> int:
```

Zero-based graph identifier within the integrand.

name
Kind: Getter

```
def name(self) -> str:
```

Persistent graph name.

is_master
Kind: Getter

```
def is_master(self) -> bool:
```

Whether this graph is the representative graph of its group.

Exhaustive reference · Source

IntegrandGraphGroup

Graphs and shared orientation, loop-basis, threshold, and cut data in one group.

Graphs and shared orientation, loop-basis, threshold, and cut data in one group.

```
class IntegrandGraphGroup:
```

Requires features: `python_api, python_abi, python_stubgen`

group_id

Kind: Getter

```
def group_id(self) -> int:
```

Zero-based graph-group identifier within the integrand.

graphs

Kind: Getter

```
def graphs(self) -> list[IntegrandGraph]:
```

Graphs in this group, with the representative graph marked as master.

orientation_edge_ids

Kind: Getter

```
def orientation_edge_ids(self) -> list[int]:
```

Edge identifiers that establish the ordering of every orientation signature.

orientations

Kind: Getter

```
def orientations(self) -> list[IntegrandOrientation]:
```

Available causal-flow orientations for the representative graph.

loop_momentum_bases

Kind: Getter

```
def loop_momentum_bases(self) -> list[IntegrandLoopMomentumBasis]:
```

Available loop-momentum bases for the representative graph.

threshold_esurface_ids

Kind: Getter

```
def threshold_esurface_ids(self) -> list[int]:
```

Ordered identifiers of threshold E-surfaces retained for the group.

threshold_esurfaces

Kind: Getter

```
def threshold_esurfaces(self) -> list[IntegrandThresholdEsurface]:
```

Structured metadata for the retained threshold E-surfaces.

cuts

Kind: Getter

```
def cuts(self) -> list[IntegrandCut]:
```

Cross-section cuts; empty for amplitude graph groups.

Exhaustive reference · Source

IntegrandInfo

Structured description of a generated integrand and its evaluation backend.

Structured description of a generated integrand and its evaluation backend.

class IntegrandInfo:

Requires features: python_api, python_abi, python_stubgen

process_id

Kind: Getter

def process_id(self) -> int:

Numeric identifier of the process that owns the integrand.

process_name

Kind: Getter

def process_name(self) -> str:

Persistent folder name of the owning process.

integrand_name

Kind: Getter

def integrand_name(self) -> str:

Canonical name of the resolved integrand.

kind

Kind: Getter

def kind(self) -> str:

``"amplitude"`` or ``"cross section"``.

generation_backend

Kind: Getter

def generation_backend(self) -> str:

Compilation backend frozen into the generated integrand.

generation_compile_options

Kind: Getter

def generation_compile_options(self) -> Optional[str]:

Backend-specific compilation options, when configured.

active_f64_backend

Kind: Getter

def active_f64_backend(self) -> str:

Floating-point evaluator backend currently active for f64 evaluation.

graph_count

Kind: Getter

```
def graph_count(self) -> int:
```

Number of amplitude graphs or cross-section supergraphs.

graph_group_count
Kind: Getter

```
def graph_group_count(self) -> int:
```

Number of graph groups in the generated integrand.

record_size_bytes
Kind: Getter

```
def record_size_bytes(self) -> int:
```

Serialized integrand-record size in bytes, excluding its compiled evaluator.

graph_groups
Kind: Getter

```
def graph_groups(self) -> list[IntegrandGraphGroup]:
```

Per-group graph, orientation, basis, threshold, and cut metadata.

Exhaustive reference · Source

IntegrandLoopMomentumBasis

Loop-momentum basis and optional multichannel identifier for a graph group.

Loop-momentum basis and optional multichannel identifier for a graph group.

```
class IntegrandLoopMomentumBasis:
```

Requires features: `python_api, python_abi, python_stubgen`

basis_id
Kind: Getter

```
def basis_id(self) -> int:
```

Zero-based loop-momentum-basis identifier on the representative graph.

channel_id
Kind: Getter

```
def channel_id(self) -> Optional[int]:
```

Effective multichannel identifier, or ``None`` when this basis is not sampled.

edge_ids
Kind: Getter

```
def edge_ids(self) -> list[int]:
```

Edge identifiers whose momenta form the basis.

matches_generation_basis
Kind: Getter

```
def matches_generation_basis(self) -> bool:
```

Whether this is the loop-momentum basis used during integrand generation.

Exhaustive reference · Source

IntegrandOrientation

Edge-direction signature for one causal-flow orientation.

Edge-direction signature for one causal-flow orientation.

```
class IntegrandOrientation:
```

Requires features: `python_api, python_abi, python_stubgen`

orientation_id

Kind: Getter

```
def orientation_id(self) -> int:
```

Zero-based orientation identifier within the graph group.

signature

Kind: Getter

```
def signature(self) -> list[int]:
```

Direction per ``IntegrandGraphGroup.orientation_edge_ids``: ``1``, ``-1``, or ``0``.

Exhaustive reference · Source

IntegrandThresholdEsurface

Threshold E-surface metadata and the cuts on which it remains active.

Threshold E-surface metadata and the cuts on which it remains active.

```
class IntegrandThresholdEsurface:
```

Requires features: `python_api, python_abi, python_stubgen`

esurface_id

Kind: Getter

```
def esurface_id(self) -> int:
```

E-surface identifier within the graph group.

representative_graph_id

Kind: Getter

```
def representative_graph_id(self) -> int:
```

Graph whose local E-surface supplies ``edge_ids``.

edge_ids

Kind: Getter

```
def edge_ids(self) -> list[int]:
```

Edge identifiers contributing energies to this E-surface.

classification

Kind: Getter

```
def classification(self) -> Optional[str]:
```

Kinematic existence class, or ``None`` when no static classification was computed.

active_cuts

Kind: Getter

```
def active_cuts(self) -> list[IntegrandActiveThresholdCut]:
```

Cuts on which this E-surface remains viable.

Exhaustive reference · Source

SampleEvaluationResult

Numerical value, metadata, stability records, and generated events for one sample.

Numerical value, metadata, stability records, and generated events for one sample.

class SampleEvaluationResult:

Requires features: python_api, python_abi, python_stubgen

integrand_result

Kind: Getter

def integrand_result(self) -> complex:

Complex integrand value before applying the parameterization Jacobian.

integrator_weight

Kind: Getter

def integrator_weight(self) -> float:

Monte Carlo weight supplied by the integrator, excluding the parameterization Jacobian.

generated_event_count

Kind: Getter

def generated_event_count(self) -> Optional[int]:

Number of candidate events generated, or ``None`` when metadata was omitted.

accepted_event_count

Kind: Getter

def accepted_event_count(self) -> Optional[int]:

Number of generated events accepted by selectors, or ``None`` without metadata.

event_processing_time_seconds

Kind: Getter

def event_processing_time_seconds(self) -> Optional[float]:

Time spent building and selecting events, in seconds, or ``None`` without metadata.

parameterization_jacobian

Kind: Getter

def parameterization_jacobian(self) -> Optional[float]:

Sample parameterization Jacobian, or ``None`` when no Jacobian was supplied.

is_nan

Kind: Getter

def is_nan(self) -> Optional[bool]:

Whether evaluation metadata flagged a non-finite result, or ``None`` without metadata.

stability_results

Kind: Getter

```
def stability_results(self) -> Optional[list[StabilityResult]]:
```

Per-precision stability attempts, or ``None`` when metadata was omitted.

event_groups

Kind: Getter

```
def event_groups(self) -> list[EventGroup]:
```

Correlated event groups produced while evaluating this sample.

__str__

Kind: Method

```
def __str__(self) -> str:
```

Return a human-readable evaluation summary.

__repr__

Kind: Method

```
def __repr__(self) -> str:
```

Return the human-readable evaluation summary.

Exhaustive reference · Source

SettingsValue

Read-only, detached view over serialized GammaLoop settings.

Read-only, detached view over serialized GammaLoop settings.

```
class SettingsValue:
```

Requires features: `python_api, python_abi, python_stubgen`

path

Kind: Getter

```
def path(self) -> str:
```

Dot-separated location of this value within the settings tree.

kind

Kind: Getter

```
def kind(self) -> str:
```

JSON value kind: ``object``, ``array``, ``string``, ``number``, ``boolean``, or ``null``.

is_object

Kind: Method

```
def is_object(self) -> bool:
```

Return whether this value is an object with named children.

is_array

Kind: Method

```
def is_array(self) -> bool:
```

Return whether this value is an array with indexed children.

is_scalar

Kind: Method

```
def is_scalar(self) -> bool:
```

Return whether this value is a scalar JSON value, including ``null``.

keys

Kind: Method

```
def keys(self) -> list[str]:
```

Return this object's child keys in lexical order.

```
Raises
------
TypeError
    If this value is not an object.
```

get

Kind: Method

```
def get(self, path: str) -> Any:
```

Resolve a dot-separated descendant path.

```
An empty path returns this value. Objects become ``SettingsValue`` views,
while scalar values become their ordinary Python equivalents.

Parameters
----------
path : str
    Relative path such as ``"sampling.parameterization"``.

Raises
------
KeyError
    If the path does not exist below this value.
```

path

Kind: Parameter

```
str
```

Relative path such as ``"sampling.parameterization"``.

to_dict

Kind: Method

```
def to_dict(self) -> Any:
```

Convert the complete detached value tree to ordinary Python containers and scalars.

__getattr__

Kind: Method

```
def __getattr__(self, name: str) -> Any:
```

Resolve an object child by attribute name.

```
Parameters
----------
name : str
    Immediate child key to resolve.
```

name

Kind: Parameter

str

Immediate child key to resolve.

__dir__

Kind: Method

```
def __dir__(self) -> list[str]:
```

Return object-child names in lexical order, or an empty list for other kinds.

__len__

Kind: Method

```
def __len__(self) -> int:
```

Return the number of object or array children.

__str__

Kind: Method

```
def __str__(self) -> str:
```

Return the detached settings value as pretty-printed JSON.

__repr__

Kind: Method

```
def __repr__(self) -> str:
```

Return a compact representation containing this value's path and kind.

__getitem__

Kind: Method

```
def __getitem__(self, key: Any, /) -> Any:
```

Resolve a relative string path or array index.

```
Parameters
----------
key : str or int
    Descendant path for an object, or zero-based index for an array.
```

key

Kind: Parameter

Any

Descendant path for an object, or zero-based index for an array.

Exhaustive reference · Source

StabilityResult

Outcome and cost of one numerical-stability precision level.

Outcome and cost of one numerical-stability precision level.

class StabilityResult:

Requires features: python_api, python_abi, python_stubgen

precision

Kind: Getter

def precision(self) -> str:

Numerical precision used for this stability level.

estimated_relative_accuracy

Kind: Getter

def estimated_relative_accuracy(self) -> Optional[float]:

Estimated relative accuracy, or ``None`` when it could not be estimated.

status

Kind: Getter

def status(self) -> str:

Human-readable stability outcome and number of rotated samples.

sample_count

Kind: Getter

def sample_count(self) -> int:

Number of rotated samples evaluated at this precision level.

total_time_seconds

Kind: Getter

def total_time_seconds(self) -> float:

Total wall-clock time spent at this precision level, in seconds.

Exhaustive reference · Source

atom_to_canonical_string

Parse a Symbolica expression and return its canonical string form.

Parse a Symbolica expression and return its canonical string form.

```
Parameters
----------
atom_str : str
    Symbolica expression to parse and normalize.

Returns
-------
str
    The parsed expression in canonical string form.

Raises
------
Exception
    If ``atom_str`` is not a valid Symbolica expression.
```

```
Examples
--------
Normalize a symbolic expression before comparing or storing it:

```python
atom_to_canonical_string("x + x")
```
```

```
def atom_to_canonical_string(atom_str: str) -> str:
```

Requires features: `python_api, python_abi, python_stubgen`

Exhaustive reference · Source

`evaluate_graph_overall_factor`

```
Evaluate a graph's symbolic overall factor and return its canonical form.
```

```
Evaluate a graph's symbolic overall factor and return its canonical form.

Parameters
----------
overall_factor : str
    Symbolica expression used as the graph's overall factor.

Returns
-------
str
    The evaluated factor in canonical Symbolica form.

Raises
------
Exception
    If the expression cannot be parsed or evaluated.

Examples
--------
Evaluate the symmetry and fermion-loop factors attached to a graph:

```python
evaluate_graph_overall_factor(
    "AutG(2)^-1*InternalFermionLoopSign(-1)"
)
```
```

```
def evaluate_graph_overall_factor(overall_factor: str) -> str:
```

Requires features: `python_api, python_abi, python_stubgen`

Exhaustive reference · Source

`to_dots`

```
Rewrite a Symbolica tensor expression into Idenso dot-product notation.
```

```
Rewrite a Symbolica tensor expression into Idenso dot-product notation.

Parameters
----------
atom_str : str
    Symbolica tensor expression whose repeated indices encode contractions.
```

```
Returns
-------
str
    The equivalent expression using Idenso dot-product notation.

Raises
------
Exception
    If ``atom_str`` cannot be parsed or converted.

Examples
--------
Rewrite a contraction before passing it to an Idenso workflow:

```python
to_dots("p(mu) * q(mu)")
```
```

```
def to_dots(atom_str: str) -> str:
```

Requires features: `python_api, python_abi, python_stubgen`

Exhaustive reference · Source

CLI commands and settings

Commands and settings available in this version are listed below.

Command tree

Command	Description
`gammaLoop`	An implementation of the Local Unitarity method for computing differential collider cross-sections.
`gammaLoop display`	Inspect models, processes, integrands, settings, and named session data
`gammaLoop display model`	Show model particles, parameters, couplings, and vertices for the selected context
`gammaLoop display processes`	List processes currently stored in the active state
`gammaLoop display integrand`	Show generated integrands, graph groups, resources, and selected detail tables
`gammaLoop display quantities`	List configured named quantities, or inspect one quantity in a selected process
`gammaLoop display observables`	List configured observables, or inspect one observable in a selected process
`gammaLoop display selectors`	List configured event selectors, or inspect one selector in a selected process
`gammaLoop display command_block`	Show one stored command block, or list every block when no name is supplied
`gammaLoop display settings`	Show effective global, default-runtime, or process-specific settings
`gammaLoop display settings global`	Show effective global generation, logging, and parallelization settings
`gammaLoop display settings default-runtime`	Show the default runtime settings inherited by newly generated integrands
`gammaLoop display settings process`	Show effective settings for one generated process or integrand
`gammaLoop duplicate`	Copy generated data to a new process or integrand name
`gammaLoop duplicate integrand`	Copy one generated integrand and its process context under new names
`gammaLoop set`	Change global, runtime, model, or process settings
`gammaLoop set base-dir`	Set the base directory for gammaloop state and logs
`gammaLoop set global`	Change global generation, logging, and parallelism settings
`gammaLoop set global file`	Merge a TOML file into the current global or default-runtime settings
`gammaLoop set global string`	Load settings from TOML content passed as a CLI string
`gammaLoop set global kv`	Merge one or more dotted `KEY=VALUE` assignments into the current settings

Command	Description
gammaLoop set global defaults	Sync process runtime settings from the current default runtime settings
gammaLoop set global stored	Reload the corresponding settings from the active state's stored file
gammaLoop set default-runtime	Change the runtime settings inherited by newly generated integrands
gammaLoop set default-runtime file	Merge a TOML file into the current global or default-runtime settings
gammaLoop set default-runtime string	Load settings from TOML content passed as a CLI string
gammaLoop set default-runtime kv	Merge one or more dotted `KEY=VALUE` assignments into the current settings
gammaLoop set default-runtime defaults	Sync process runtime settings from the current default runtime settings
gammaLoop set default-runtime stored	Reload the corresponding settings from the active state's stored file
gammaLoop set model	Assign model parameters in the selected context or reset them to defaults
gammaLoop set process	Change runtime settings or named analysis objects for one process or integrand
gammaLoop set process file	Merge a TOML file into the selected process or integrand settings
gammaLoop set process string	Load settings from TOML content passed as a CLI string
gammaLoop set process kv	Merge one or more dotted `KEY=VALUE` assignments into the selected settings
gammaLoop set process defaults	Sync process runtime settings from the current default runtime settings
gammaLoop set process stored	Reload the selected process or integrand settings from its stored file
gammaLoop set process add	Add a named quantity/observable/selector
gammaLoop set process add quantity	Add a named derived quantity to the selected process settings
gammaLoop set process add observable	Add a named observable to the selected process settings
gammaLoop set process add selector	Add a named event selector to the selected process settings
gammaLoop set process update	Update a named quantity/observable/selector
gammaLoop set process update quantity	Update fields of an existing named quantity
gammaLoop set process update observable	Update fields of an existing named observable
gammaLoop set process update selector	Update fields of an existing named selector
gammaLoop set process remove	Remove a named quantity/observable/selector

Command	Description
gammaLoop set process remove quantity	Remove a named quantity from the selected process settings
gammaLoop set process remove observable	Remove a named observable from the selected process settings
gammaLoop set process remove selector	Remove a named selector from the selected process settings
gammaLoop import	Import models or previously generated graphs into the active state
gammaLoop import model	Load a UFO model and make it the active model for subsequent generation
gammaLoop import graphs	Load serialized graph data into a new or existing process
gammaLoop save	Export graph data, standalone evaluators, schemas, or the current state
gammaLoop save dot	Export generated graphs as DOT files and drawing templates
gammaLoop save uv-forest	Export the ultraviolet forest for selected generated graphs
gammaLoop save standalone	Export an integrand as a standalone evaluator or serialized data file
gammaLoop save state	Persist the active state, settings, and replayable run history
gammaLoop save schema	Regenerate the JSON schema files for CLI and runtime configuration
gammaLoop run	Execute a stored command block or an inline command list
gammaLoop start_commands_block	Begin recording subsequent commands under a reusable block name
gammaLoop finish_commands_block	Stop recording commands and store the active command block
gammaLoop remove	Remove generated processes from the active state
gammaLoop remove processes	Remove one integrand, one process, or all processes selected by the supplied target
gammaLoop integrate	Integrate one or more generated process integrands
gammaLoop generate	Generate cross-section or amplitude integrands from a process specification
gammaLoop generate xs	Generate a cross-section integrand through the forward-scattering construction
gammaLoop generate amp	Generate an amplitude integrand for the supplied process specification
gammaLoop generate existing	Create another integrand from an existing process without regenerating its graphs
gammaLoop select	Select graph groups and write the selection to a new process or integrand

Command	Description
gammaLoop quit	End the interactive GammaLoop session without executing further commands
gammaLoop inspect	Inspect integrand contributions at one phase-space point or momentum configuration
gammaLoop approach	Sample an integrand while approaching a phase-space point along selected scaling axes
gammaLoop evaluate	Evaluate a vacuum amplitude analytically or numerically with Vakint
gammaLoop renormalize	Compute and export ultraviolet renormalization contributions for an amplitude
gammaLoop bench	Measure raw integrand evaluation throughput over random samples and selected worker counts
gammaLoop profile	Profile ultraviolet or infrared scaling limits of a generated integrand
gammaLoop profile ultra-violet	Sample ultraviolet scaling rays and report the fitted large-momentum behavior
gammaLoop profile bulk	Probe multiple infrared scaling limits and report their fitted behavior in one run
gammaLoop batch	Evaluate a process-definition file against an HPC batch input and write named results
gammaLoop !	Run an operating-system shell command from the interactive GammaLoop session

Settings

Path	Type	Default
cli.global.display_directive	string	"info"
cli.global.generation.compile.compilation_mode	CompilationMode	"symjit"
cli.global.generation.compile.compiler	string	"g++"
cli.global.generation.compile.custom	array<string>	[]
cli.global.generation.compile.fast_math	boolean	true
cli.global.generation.compile.optimization_level	OptimizationLevel	"O3"
cli.global.generation.compile.unsafe_math	boolean	true
cli.global.generation.evaluator.active_level	integer	0
cli.global.generation.evaluator.compile	boolean	false
cli.global.generation.evaluator.coupling_reruns	integer | null	no serialized default
cli.global.generation.evaluator.direct_translation	boolean	true
cli.global.generation.evaluator.dual_cff	boolean	false
cli.global.generation.evaluator.dual_ltd	boolean	false
cli.global.generation.evaluator.horner_iterations	integer	1
cli.global.generation.evaluator.iterative_orientation_optimization	boolean	true
cli.global.generation.evaluator.max_common_pair_cache_entries	integer	1000000
cli.global.generation.evaluator.max_common_pair_distance	integer	1000
cli.global.generation.evaluator.max_horner_scheme_variables	integer	500

Path	Type	Default
cli.global.generation.evaluator.n_cores	integer	1
cli.global.generation.evaluator.spenso_execution_mode	tuple<ExecutionMode, ContractionMode>	["Sequential","MinResultRank"]
cli.global.generation.evaluator.store_atom	boolean	false
cli.global.generation.evaluator.summed	boolean	false
cli.global.generation.evaluator.summed_function_map	boolean	false
cli.global.generation.evaluator.tensor_network_contraction_order	TensorNetworkContractionOrder	"[illegible]_aware"
cli.global.generation.evaluator.verbose	boolean	false
cli.global.generation.feyngen.gamma_simplification_closure_check	boolean	false
cli.global.generation.force_cuts	array<array<string>>	[]
cli.global.generation.orientation_pattern	null | string	no serialized default
cli.global.generation.override_lmb_heuristics	boolean	false
cli.global.generation.threshold_subtraction.assume_positive_external_energies	boolean	true
cli.global.generation.threshold_subtraction.check_esurface_at_generation	boolean	false
cli.global.generation.threshold_subtraction.disable_integrated_ct	boolean	false
cli.global.generation.threshold_subtraction.enable_thresholds	boolean	true
cli.global.generation.threshold_subtraction.esurface_existence_threshold	number	1e-7
cli.global.generation.threshold_subtraction.skip_thresholds_that_are_cuts	boolean	true
cli.global.generation.tropical_subgraph_table.disable_tropical_generation	boolean	false
cli.global.generation.tropical_subgraph_table.panic_on_fail	boolean	false
cli.global.generation.tropical_subgraph_table.target_omega	number	1.0
cli.global.generation.uv.add_marker	boolean	false
cli.global.generation.uv.final_integrand	FinalIntegrandDimension	"ThreeD"
cli.global.generation.uv.generate_integrated	boolean	true
cli.global.generation.uv.inner_products	boolean	true
cli.global.generation.uv.keep_marker	boolean	true
cli.global.generation.uv.orchestrator	UVOrchestrator	"hedge_poset"
cli.global.generation.uv.renormalization_prescription.log_divergent	ApproximationType	"MUV"
cli.global.generation.uv.renormalization_prescription.massive_power_divergent	ApproximationType	"MUV"
cli.global.generation.uv.renormalization_prescription.massless_power_divergent	ApproximationType	"MUV"
cli.global.generation.uv.renormalization_prescription.overrides	array<CTRenormalizationRule>	[]
cli.global.generation.uv.renormalization_prescription.overrides[].ct_identifier.external_pdg_set	array<integer>	no serialized default
cli.global.generation.uv.renormalization_prescription.overrides[].ct_identifier.internal_pdg_set	array | null	no serialized default
cli.global.generation.uv.renormalization_prescription.overrides[].prescription	ApproximationType	no serialized default
cli.global.generation.uv.softct	boolean	true
cli.global.generation.uv.subtract_uv	boolean	true
cli.global.generation.uv.vakint.additional_normalization	string	"1"
cli.global.generation.uv.vakint.alphaloop.susbstitute_masters	boolean	true
cli.global.generation.uv.vakint.clean_tmp_dir	boolean	true
cli.global.generation.uv.vakint.evaluation_methods	array<string>	["alphaloop","matad","fmft"]
cli.global.generation.uv.vakint.fmft.expand_masters	boolean	true

Path	Type	Default
cli.global.generation.uv.vakint.form.use_substitute_masters	boolean	true
cli.global.generation.uv.vakint.form_exe_path	string	"form"
cli.global.generation.uv.vakint.mata.direct_numerical_substition	boolean	true
cli.global.generation.uv.vakint.mata.expand_masters	boolean	true
cli.global.generation.uv.vakint.mata.substitute_hpls	boolean	true
cli.global.generation.uv.vakint.mata.substitute_masters	boolean	true
cli.global.generation.uv.vakint.normalization	string	"MSbar"
cli.global.generation.uv.vakint.pysecdec.max_n_evals	integer	1000000000000
cli.global.generation.uv.vakint.pysecdec.min_n_evals	integer	10000
cli.global.generation.uv.vakint.pysecdec.quiet	boolean	true
cli.global.generation.uv.vakint.pysecdec.relative_precision	number	1e-7
cli.global.generation.uv.vakint.pysecdec.reuse_existing_output	string | null	no serialized default
cli.global.generation.uv.vakint.python_exe_path	string	"python3"
cli.global.generation.uv.vakint.run_time_decimal_precision	integer	100
cli.global.generation.uv.vakint.temporary_directory	string | null	no serialized default
cli.global.generation.vector_polarization_convention	VectorPolarizationConvention	"SingleAxial"
cli.global.log_style.full_line_source	boolean	false
cli.global.log_style.include_fields	boolean	false
cli.global.log_style.log_format	LogFormat	"Long"
cli.global.log_style.short_timestamp	boolean	false
cli.global.logfile_directive	string	"off"
cli.global.n_cores.compile	integer	1
cli.global.n_cores.feyngen	integer	1
cli.global.n_cores.generate	integer	1
cli.global.n_cores.integrate	integer	1
cli.override_state	boolean	false
cli.state.folder	string	"./gammaloop_state"
cli.state.name	string | null	""
cli.try_strings	boolean	true
runtime.general.additional_param_values	array<number>	[]
runtime.general.debug_cache	boolean	false
runtime.general.disable_flux_factor	boolean	false
runtime.general.enable_cache	boolean	false
runtime.general.evaluator_method	EvaluatorMethod	"SingleParametric"
runtime.general.generate_events	boolean	false
runtime.general.integral_unit	IntegralUnit	"auto"
runtime.general.load_compiled_cff	boolean	false
runtime.general.m_uv	number	1000.0
runtime.general.mu_r	number	1000.0
runtime.general.orientation_pat.pat	null | string	no serialized default

Path	Type	Default
runtime.general.renormalization_local_cal	boolean	false
runtime.general.store_additional_weights_in	boolean	false
runtime.general.use_ltd	boolean	false
runtime.h_function.enabled_dampening	boolean	true
runtime.h_function.function	HFunction	"poly_exponential"
runtime.h_function.power	integer | null	no serialized default
runtime.h_function.sigma	number	1.0
runtime.integrand.h_function.enabled_dampening	boolean	no serialized default
runtime.integrand.h_function.function	HFunction	no serialized default
runtime.integrand.h_function.power	integer | null	no serialized default
runtime.integrand.h_function.sigma	number	no serialized default
runtime.integrand.n_3d_momenta	integer	no serialized default
runtime.integrator.bin_number_evolution	array | null	no serialized default
runtime.integrator.continuous_dim_learning_rate	number	1.0
runtime.integrator.discrete_dim_learning_rate	number	1.0
runtime.integrator.integrated_phase	IntegratedPhase	"real"
runtime.integrator.max_prob_ratio	number	30.0
runtime.integrator.min_samples_for_update	integer	1000
runtime.integrator.n_bins	integer	64
runtime.integrator.n_increase	integer	10000
runtime.integrator.n_max	integer	10000000000
runtime.integrator.n_start	integer	100000
runtime.integrator.observables_output_format	array<ObservableFileFormat>	["lhem"]
runtime.integrator.seed	integer	69
runtime.integrator.show_max_wgt_info	boolean	true
runtime.integrator.target_absolute_accuracy	number | null	no serialized default
runtime.integrator.target_relative_accuracy	number | null	no serialized default
runtime.integrator.train_on_avg	boolean	false
runtime.kinematics.e_cm	number	64.0
runtime.kinematics.externals.data.helicities	array<integer | string>	[]
runtime.kinematics.externals.data.improvement_settings	improvement_settings	log_12_large_deformation_check
runtime.kinematics.externals.data.improvement_settings.mode	ImprovementSettingsMode	"none"
runtime.kinematics.externals.data.improvement_settings.only_warn_on_large_deformation	boolean	false
runtime.kinematics.externals.data.momenta	array<MomentaStep | array<number>>	[]
runtime.kinematics.externals.type	string	"constant"
runtime.model	object	{}
runtime.observables	object	{}
runtime.quantities	object	{}
runtime.sampling.alpha	number	3.0

Path	Type	Default
runtime.sampling.b	number	1.0
runtime.sampling.coordinate_system	CoordinateSystem	"spherical"
runtime.sampling.graph_names	array<string>	[]
runtime.sampling.graphs	SumMode	"summed"
runtime.sampling.lmb_basis_ids	object	no serialized default
runtime.sampling.lmb_channel_weight	LmbChannelWeight	"ose"
runtime.sampling.lmb_channels	SumMode	"summed"
runtime.sampling.lmb_multichanneling	boolean	false
runtime.sampling.mapping	ParameterizationMapping	"linear"
runtime.sampling.orientations	SumMode	"summed"
runtime.sampling.power	number	1.0
runtime.selectors	object	{}
runtime.stability.check_on_norm	boolean	true
runtime.stability.escalate_if_exact_zero	boolean	false
runtime.stability.levels	array<StabilityLevelSetting>	[{"escalate_for_large_weight_threshold":0.9,"pr {"escalate_for_large_weight_threshold":-1.0,"pr {"escalate_for_large_weight_threshold":-1.0,"pr
runtime.stability.levels[].escalate_for_large_weight_threshold	number	no serialized default
runtime.stability.levels[].precision	Precision	no serialized default
runtime.stability.levels[].required_precision_for_im	number	no serialized default
runtime.stability.levels[].required_precision_for_re	number	no serialized default
runtime.stability.loop_momenta_norm_escalation_factor	number	1.0
runtime.stability.recording.recorded_stability_levels	boolean	no serialized default
runtime.stability.recording.recorded_loop_momenta_escalation	boolean	no serialized default
runtime.stability.recording.recorded_results	boolean	no serialized default
runtime.stability.rotation_axis	array<object>	[{"alpha":0.1,"beta":0.2,"gamma":0.3,"type":"eu
runtime.stability.rotation_axis[].alpha	number	no serialized default
runtime.stability.rotation_axis[].beta	number	no serialized default
runtime.stability.rotation_axis[].gamma	number	no serialized default
runtime.stability.rotation_axis[].type	string	no serialized default
runtime.subtraction.disable_threshold_subtraction	boolean	false
runtime.subtraction.esurface_existence_threshold	number	1e-7
runtime.subtraction.integrated_ct_settings.range.h_function	settings.range.h_function_settings.enabled_dampening
runtime.subtraction.integrated_ct_settings.range.h_function	settings.range.h_function_settings.function
runtime.subtraction.integrated_ct_settings.range.h_function	settings.range.h_function_settings.power
runtime.subtraction.integrated_ct_settings.range.h_function	settings.range.h_function_settings.sigma
runtime.subtraction.integrated_ct_settings.range.type	string	"infinite"
runtime.subtraction.local_ct_settings.uv_localisation.dy	boolean	false
runtime.subtraction.local_ct_settings.uv_localisation.se	boolean	false
runtime.subtraction.local_ct_settings.uv_localisation.gaussian_width	number	1.0
runtime.subtraction.local_ct_settings.uv_localisation.sliver_width	number	10.0

Path	Type	Default
runtime.subtraction.overlap_settings.check_global_center	boolean	true
runtime.subtraction.overlap_settings.forced_global_center	array | null	not serialized default
runtime.subtraction.overlap_settings.keep_origin	boolean	true
runtime.subtraction.overlap_settings.keep_origin_all	boolean	false
runtime.subtraction.radial_root_residual_tolerance	number	64.0

Version information

Save these identifiers with published results and include them in issue reports so that collaborators can reproduce the same software environment.

- **Documentation channel:** `latest`
- **Documentation route:** `/products/gammaloop/latest/`
- **Source revision:** `e51747446aa724c3ffac5c42c4103d090c09c6b0`

Package versions and enabled features

- `gammaloop/gammalooprs` — `0.3.4` (`no optional features`)
- `gammaloop/gammaloop-api` — `0.1.0` (`features: cli`)
- `gammaloop/gammaloop` — `0.3.4` (`features: python_api, python_abi, python_stubgen`)
- `linnet/linnet` — `0.17.0` (`features: nodestore-vec, serde, bincode, drawing, symbolica`)
- `linnet/linnet-py` — `0.1.0` (`features: extension-module, abi3-py310`)
- `spenso/spenso` — `0.6.0` (`features: shadowing`)
- `spenso/spenso-macros` — `0.3.2` (`features: shadowing`)
- `spenso/spenso-hep-lib` — `0.1.7` (`no optional features`)
- `spenso/spynso3` — `0.1.2` (`features: python_stubgen`)
- `idenso/idenso` — `0.3.0` (`features: bincode, reference-cases`)
- `idenso/idenso` — `0.3.0` (`features: python, python_stubgen`)
- `vakint/vakint` — `0.1.2` (`no optional features`)
- `vakint/vakint` — `0.1.2` (`features: symbolica_community_module, python_stubgen`)